TURING 图灵程序设计丛书

AMACOM

Debugging

The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems

调试九法

软硬件错误的排查之道

[美] David J. Agans 著

赵俐 译





人民邮电出版社
POSTS & TELECOM PRESS

“《调试九法》道出了九项调试最佳实践，这些实践是优秀程序员的基本常识，也是普通程序员都在遵循的规则。程序员阅读本书必将会受益匪浅，而调试行家也将从书中示例得到不少启发。我们很高兴地发现需要遵守的调试规则比圣经中的戒律要少得多。”

——Rob Malda

slashdot.org网站创始人

“作者写了一本令人愉快的书，书中引入了福尔摩斯探案等有趣的实例，相当幽默风趣。我当然会为我的客户买这本书。”

——Dick Morley

PLC之父，R. Morley公司总裁

*The Technology Machine*作者

“本书明确阐释了九条永恒的、不可或缺的调试原则，它们可以帮助任何人进行任何调试。即使对于那些最有经验的工程师，本书也不无裨益。”

——Howard Johnson博士

*High-Speed Digital Design*的作者

“本书帮助你拨开迷雾，成为一个聪明的调试者。”

——Charles Petzold

*Programming Windows*的作者

图书在版编目（C I P）数据

调试九法 ：软硬件错误的排查之道 / (美) 阿甘斯
(Agans,D.J.) 著 ; 赵俐译. -- 北京 : 人民邮电出版社,
2010.12
（图灵程序设计丛书）
书名原文: Debugging:The 9 Indispensable Rules
for Finding Even the Most Elusive Software and
Hardware Problems
ISBN 978-7-115-24057-6

Ⅰ. ①调… Ⅱ. ①阿… ②赵… Ⅲ. ①电子计算机－
调试 Ⅳ. ①TP306

中国版本图书馆CIP数据核字(2010)第209193号

内 容 提 要

本书主要介绍了调试方面的 9 条黄金法则，并结合实际的环境讲述了如何合理地运用它们。本书的内容没有针对任何平台、任何语言或者任何工具，讲述的重点是找到出错的原因并修复它们，高效地追踪和解决不易察觉的软硬件问题。

本书适合所有软硬件从业人员阅读。

图灵程序设计丛书

调试九法：软硬件错误的排查之道

◆ 著　[美] David J. Agans
译　赵　俐
责任编辑　傅志红
执行编辑　谢灵芝

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街14号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　印刷

◆ 开本：800×1000　1/16
印张：9.75
字数：147千字　2010年 12 月第 1 版
印数：3 000　2010年 12 月北京第 1 次印刷

著作权合同登记号　图字：01-2010-5547号

ISBN 978-7-115-24057-6

定价：35.00元

读者服务热线：(010)51095186　印装质量热线：(010)67129223

反盗版热线：(010)67171154

版 权 声 明

谨以此书献给我的母亲Ruth（Worsley）Agans，感谢她帮我调试Fortran程序，她借着无数杯浓咖啡的帮助，在我们的餐厅里完成了这些工作。

同时也献给我的父亲John Agans，是他教会我思考，教会我如何利用自己的常识，也教会我开朗乐观。他们的精神给了我无尽的动力。

致　　谢

本书的孕育可追溯至1981年，当时Gould公司的一组测试工程师问我是否能用一篇文档写清楚如何解决硬件产品问题。我听了之后有点不知所措，因为我们的产品是一些由上百块芯片、几个微处理器和无数通信总线组成的主板。我深知没有“魔力配方”，他们必须学会如何调试。我与Mike Bromberg（他一直是我的良师益友）讨论了这件事，我们达成共识——最起码要编写一些通用的调试规则。于是我最后编写了“Ten Debugging Commandments”（调试十诫），这是一页简短的调试规则，测试小组很快把它贴到了墙上。几年后，这份清单已被压缩为一条规则，而且被推广应用到很多软件和系统，但它仍然是本书的核心。因此感谢Mike和那些提出这个请求的基层技术人员。

感谢Doug Currie、Scott Ross、Glen Dash、Dick Morley、Mike Greenberg、Cos Fricano、John Aylesworth（原来提出请求的技术人员之一）、Bob DeSimone和Warren Bayek，他们使得那些富有挑战性的工作变得充满乐趣。一直以来，我都很高兴能够为他们工作并与他们并肩作战，他们带给我无数灵感，帮助我培养了调试技巧，也教会了我幽默。还要感谢三位追求卓越并为我的学习过程带来乐趣的老师，他们是Nick Menutti（虽然这不是诺贝尔奖，但这里让我奉上对你的赞誉）、Ray Fields和Professor Francis F. Lee。还要感谢我一直未曾谋面的几位作者，他们的书对我的创作产生了巨大影响，他们是William Strunk Jr和E. B. White（著有*The Elements of Style*）、Jeff Herman和Deborah Adams（著有*Write the Perfect Book Proposal*）。

感谢包容我28年之久的夏日垒球球队Delt Dawgs，感谢朋友们的审阅和帮助。Charlie Seddon详细审阅了本书并给出很多有益的评论，Bob Siedensticker也审阅了本书并为我提供了案例、主题建议和出版建议，对此我感激不尽。还有几位当时我还不认识的朋友审阅了本书并给我寄来他们的意见，本书的出版离不开他们的帮助。他们是Warren Bayek和Charlie Seddon（上面提到过）、Dick Riley、Bob Oakes、Dave Miller和Terry Simkin教授，感谢他们付出的时间和提出的意见。

感谢Sesame工作室的Tom和Ray Magliozzi。（“侃车[①]”节目的主持人Click和Clack，抑或是Clack和Click[②]？）另外感谢Steve Martin允许我使用他们的故事和笑话，还要感谢Arthur Conan Doyle（柯南道尔爵士）创造了福尔摩斯这个侦探形象，并通过他表达了如此多的妙语，此外感谢Seymour Friedel、Bob McIlvaine和我的兄弟Tom Agans为我讲述妙趣横生的案例。发现和证明这些规则离不开他们提供的例子，感谢所有案例故事的参与者，感谢“英雄和笨蛋[③]”（你们知道我说的是谁）。

与Amacom的编辑们一起工作是一段美妙的经历，而且给了我很多启发。感谢Jacquie Flynn和Jim Bessent，感谢他们的热情和中肯的建议。感谢在出版过程中作出贡献的设计者和才华横溢的人们，正是由于他们的出色工作，才使本书成功出版。

特别感谢我的代理人Jodie Rhodes，感谢他给了我初次写书的机会，让我用别具一格的方式来写这个非同寻常的主题。他了解他的市场，事实也证明他是对的。

感谢我的亲人Dick和Joan Blagbrough给予的支持、鼓励和大量帮助。特别感谢我的女儿Jen和Liz，我要拥抱和亲吻她们，她们非常可爱，也感谢她们给予我的信任。（还要感谢她们每天晚上在用IM聊天和玩游戏的间隙把电脑让给我用。）

最后，我要把永恒的爱和感激献给我的妻子Gail，感谢她鼓励我把这些调试规则写成一

---

① 侃车，Car Talk，美国国家公众广播电台的一档节目。（如无特殊说明，本书中的脚注均为译者注。）

② Click和Clack，上述“侃车”节目二位主持人Tom和Ray Magliozzi的昵称，指车子运行时发出的咔咔声。

③《英雄和笨蛋》，*heroes and fools*，龙枪系列小说之一，作者在这里借用此名开了个玩笑。

本书，感谢她督促我寻找代理人，感谢她给予我创作的时间和空间。也感谢她校对了大量的初稿，要知道，没有她的校对，这些书稿我甚至不敢拿给别人看。她可以用一个吸尘器点亮吊灯[1]，却完全凭借她自己点亮了我的人生。

Dave Agans

2002年6月

① 第13章中的案例故事有具体的解释。

译　者　序

有人说调试是一门艺术，这不无道理，但本书作者认为它并不仅仅是艺术，更多的是科学，调试人员也不仅仅是艺术家，还是科学工作者。遵循本书所讲的9条规则，就可以把调试艺术转化为科学。

本书翻译到一半的时候，我已经钦佩不已。它绝对称得上调试领域的经典之作，但显然，在某种程度上它并没有引起国内业界的注意。常言道“千里马常有，而伯乐不常有”，虽然用千里马来形容一本书多少有些不恰当，但我确实觉得本书被埋没了，我想我们应该感谢人民邮电出版社图灵公司，把这样一本好书发掘出来，让国人有机会分享这位拥有二十多年实践经验的调试高手的知识和经验。

把书写厚了容易，写薄了却难，我想这一点大家都会认同。作者正是用这么薄薄的一本书讲述了适用于软件、硬件、工程领域的9条基本调试规则。这些规则甚至还适用于我们的日常生活，例如解决汽车和房屋问题。仔细揣摩，我们会学到不少生活知识，这也是阅读本书的一个额外的好处。

本书就像是一碗心灵鸡汤，也像是一坛陈年佳酿，书中所举的一些案例散发着古朴的气息。虽然我没有怀旧情节，但仍感到亲切而自然，有那么一刻，我与作者灵犀相通，仿佛他就站在那里，正在向我微笑，与我倾谈。

作者是个福尔摩斯迷，我想这是不是与他的职业生涯有关呢？在bug面前，他就是一名侦探。每章的开头都会引用福尔摩斯的一句名言，暗合该章的主题，也为本书平添了一层神

秘的色彩。我想要是把“神探狄仁杰”的故事讲给他听，他也一定会非常感兴趣，尽管他可能连狄仁杰是谁都不知道。

在本书翻译的过程中，有些地方我思索良久，有些则需要查询一些相关知识。我觉得提供一点背景资料会有助于理解，因此加了一些脚注，对于那些不甚了解相关背景或知识的人，可能会有一点帮助作用，但有些读者可能很熟悉硬件、软件和工程领域，如果这些内容都是你所熟知的，那么请恕我赘述之过。

最后，本书在翻译的过程中得到了人民邮电出版社图灵公司编辑们的精心指导和宝贵意见，帮我纠正了翻译中的很多错误，使我受益匪浅。由于水平有限，难免还会留有一些错误，恳请读者批评指正。

目　　录

第 1 章　简介 ………………………… 1

1.1　本书如何教会你调试 ………………… 1

1.2　这些规则都很显而易见 ……………… 2

1.3　本书适用于任何人 …………………… 3

1.4　本书可用于调试各种问题 …………… 3

1.5　本书的主旨不在预防、保证或筛选 …… 4

1.6　调试不仅仅是故障检修 ……………… 5

1.7　有关案例故事 ………………………… 6

1.8　精彩内容，即将上演 ………………… 6

第 2 章　总体规则 ……………………… 8

第 3 章　理解系统 ……………………… 10

3.1　阅读手册 ……………………………… 12

3.2　逐字逐句阅读整个手册 ……………… 13

3.3　知道什么是正常的 …………………… 15

3.4　知道工作流程 ………………………… 16

3.5　了解你的工具 ………………………… 17

3.6　查阅手册 ……………………………… 18

3.7　小结 …………………………………… 20

第 4 章　制造失败 ……………………… 21

4.1　制造失败 ……………………………… 24

4.2　从头开始 ……………………………… 24

4.3　引发失败 ……………………………… 25

4.4　不要模拟失败 ………………………… 25

4.5　如何处理间歇性 bug ………………… 27

4.6　如果做了所有尝试之后问题仍然间歇性发生 ……………………………… 29

4.6.1　仔细观察失败 ……………………… 29

4.6.2　不要盲目相信统计数据 …………… 30

4.6.3　是已修复 bug，还是仅仅由于运气好，它不再发生了 ………………… 31

4.7　“那不可能发生” ……………………… 33

4.8　永远不要丢掉调试工具 ……………… 34

4.9　小结 …………………………………… 36

第 5 章　不要想，而要看 ……………… 37

5.1　观察失败 ……………………………… 41

5.2　查看细节 ……………………………… 43

5.3　问题忽隐忽现 ………………………… 46

5.4　对系统进行插装 ……………………… 46

5.4.1　设计插装工具 ……………………… 46

5.4.2　过后构建插装 ……………………… 48

5.4.3　不要害怕深入研究 ………………… 50

5.4.4　添加外部插装 ……………………… 51

5.4.5　日常生活中的插装 ………………… 51

5.5　海森堡测不准原理 …………………… 52

5.6 猜测只是为了确定搜索的重点目标 …… 53
5.7 小结 …… 54

第 6 章 分而治之 …… 55

6.1 缩小搜索范围 …… 59
6.1.1 确定范围 …… 60
6.1.2 你在哪一侧 …… 61
6.2 插入易于识别的模式 …… 62
6.3 从有问题的支路开始查找问题 …… 63
6.4 修复已知 bug …… 64
6.5 首先消除噪声干扰 …… 65
6.6 小结 …… 66

第 7 章 一次只改一个地方 …… 67

7.1 使用步枪，而不要用散弹枪 …… 69
7.2 用双手抓住黄铜杆 …… 71
7.3 一次只改变一个测试 …… 72
7.4 与正常系统进行比较 …… 73
7.5 自从上一次能够正常工作以来你更改了什么 …… 74
7.6 小结 …… 77

第 8 章 保持审计跟踪 …… 78

8.1 记下你的每步操作、顺序和结果 …… 80
8.2 魔鬼隐藏在细节中 …… 81
8.3 关联 …… 83
8.4 用于设计的审计跟踪在测试中也非常有用 …… 84
8.5 好记性不如烂笔头 …… 84
8.6 小结 …… 85

第 9 章 检查插头 …… 86

9.1 怀疑自己的假设 …… 88
9.2 从头开始检查 …… 89
9.3 对工具进行测试 …… 90
9.4 小结 …… 92

第 10 章 获得全新观点 …… 93

10.1 寻求帮助 …… 94
10.1.1 获得全新观点 …… 94
10.1.2 询问专家 …… 94
10.1.3 借鉴别人的经验 …… 95
10.2 到哪里寻求帮助 …… 96
10.3 放下面子 …… 97
10.4 报告症状，而不是理论 …… 98
10.5 小结 …… 99

第 11 章 如果你不修复 bug，它将依然存在 …… 101

11.1 检查问题确实已被修复 …… 103
11.2 检查确实是修复措施解决了问题 …… 103
11.3 bug 从来不会自己消失 …… 104
11.4 从根本上解决问题 …… 105
11.5 对过程进行修复 …… 107
11.6 小结 …… 107

第 12 章 通过一个案例讲述所有规则 …… 109

第 13 章 牛刀小试 …… 113

13.1 灯和吸尘器的故事 …… 113
13.2 大量出现的 bug …… 115
13.3 宽松的限制 …… 119
13.4 识破 bug …… 123

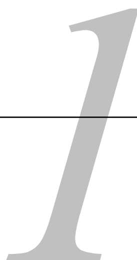
第 14 章 从帮助台得到的观点 …… 128

14.1 帮助台的限制 …… 130
14.2 规则，帮助台风格 …… 130
14.2.1 理解系统 …… 131
14.2.2 制造失败 …… 132
14.2.3 不要想，而要看 …… 132
14.2.4 分而治之 …… 134
14.2.5 一次只改一个地方 …… 134

14.2.6 保持审计跟踪 135
14.2.7 检查插头 136
14.2.8 获得全新观点 136
14.2.9 如果你不修复 bug，它将依然存在 137
14.3 小结 137
第 15 章 结束语 139
15.1 调试规则网站 139
15.2 如果你是一名工程师 139
15.3 如果你是一名经理 140
15.4 如果你是一名教师 141
15.5 小结 141

第 1 章

简　　介



“你知道，现阶段我非常忙，但我打算在晚年倾力写一本书，把所有侦探艺术都集中写到这本书里。”

——福尔摩斯，《格兰其庄园》

本书告诉你如何快速找到工作中的错误。它很短，也很有趣，因为它必须如此——如果你是一位工程师，你每天都在忙于调试，可能除了看点漫画之外就没时间读别的了。即使你不是工程师，也经常会遇到问题，这时你必须查明如何解决问题。

可能有人从来不需要做调试工作。或许你正忙着赶在公司倒闭之前把通过dot.com IPO[①]发行的股票卖出去，因而只是让你手下的人去查找问题。或许你总是很幸运，你的设计一直未发生问题，或者bug总是很容易找到（尽管这不太可能）。有可能在你和你的所有竞争对手的设计中都有一些很难查找的bug，谁能够最快地修复它们，谁就占据了优势。当你快速找到bug时，不仅能够更快地为客户提供更高质量的产品，而且也能够更早下班回家，与家人一起享受美好的时光。

因此，请把这本书放在你的床头柜上或洗手间里，两周后，你就会成为一位调试高手了。

1.1　本书如何教会你调试

为什么这样一本简短且易读的书会这么有用呢？根据我26年的系统设计和调试经验，我

① dot.com IPO，是指通过互联网公司上市募集资金。IPO，即首次公开募股（Initial Public Offering）。

发现了两件重要的事情（如果你把“从咖啡壶里倒出的第一杯咖啡含有全部的咖啡因”这样显而易见的错误也当成重要的问题，那么在你看来重要的事情就不止两件了）。

(1) 如果查找一个bug花费了大量时间，那么原因可能是忽略了某个最基本的、最重要的规则，一旦应用了那条规则，很快就会找到问题。

(2) 擅于快速调试的人已经深刻理解并应用了这些规则，而那些很难理解或使用这些规则的人则很难找到bug。

我把这些基本规则编写成一个清单，并教给其他工程师，我发现他们的调试技术和速度都提高了。这些规则的的确确起了作用。

1.2　这些规则都很显而易见

当你读到这些规则时，你可能会自言自语地说：“这些都是一些明显的规则啊。”不要着急下结论，这些规则确实都很明显（而且常常是基本的规则），但如何把它们应用于特定的问题就不总是那么显而易见了。而且不要把“显然”和“容易”混淆在一起，这些规则遵守起来并不总是那么容易，因此在解决实际问题时常常被忽略。

关键是记住并应用这些规则。如果这很明显而且容易，那么我就不必总是提醒工程师们应用这些规则，我也不必通过几十个案例故事[1]来说明不遵守这些规则将会发生什么情况。能够自如地运用这些规则的调试人员是凤毛麟角。我喜欢问求职者这样一个问题：“你调试时使用什么拇指规则[2]？”奇怪的是，很多人都回答说：“艺术。”好极了，那么让毕加索来调试我们的图像处理算法吧。事实上，利用简单和艺术的方法未必就能快速找到问题。

本书把这些“明显的”规则收集到一起，帮助你记住它们，知道它们的益处，并掌握如何运用它们，从而帮助你抵挡住“走捷径”的诱惑，因为捷径往往是陷阱。本书将把调试艺

① 案例故事，war story，原指战争故事，后泛指一些给人留下深刻印象的经历，在本书中则是作者举出的一些经典的、有代表性的案例。

② 拇指规则，英文为rule of thumb，又译为“大拇指规则 ”或“经验法则”，是一种可用于许多情况的简单的经验性的原则。

术转变为一门科学。

即使你已经是一位非常优秀的调试人员，这些规则仍然能够帮助你更上一层楼。当我把本书早期手稿拿给经验丰富的调试人员审阅时，他们不约而同地表示，本书除了教会一两个他们没有用过（但将来会用到）的规则之外，还帮助他们明确意识到了他们在不知不觉中遵守的规则。团队领导者（当然是顶尖的调试人员）指出，本书为团队提供了一种很好的沟通语言，使他们能够把技巧传授给其他成员。

1.3　本书适用于任何人

本书通篇都用“工程师”这个词来指代读者，但即使你不是工程师，这些规则对你也非常有用。当然，如果你正在查找设计中的错误，本书就更有用了，无论你是工程师、程序员、技师、客户支持代表，还是顾问。

如果你不直接参与调试工作，而是负责管理调试人员，那么也可以把这些规则传授给你的工作人员。你甚至不必理解他们所使用的系统和工具的细节，因为本书所讲的都是一些非常基本的规则，因此在读完本书后，即使你是一位“尖发经理[1]”，也能够帮助那些比你聪明得多的团队成员更快地找到问题。

如果你是一位教师，你的学生将会非常喜欢书中的案例故事，这些故事将为他们带来真实世界的体验。当他们走出校门时，将会比那些经验丰富（但没有受过调试培训）的竞争对手们更有优势。

1.4　本书可用于调试各种问题

本书具有很强的通用性，它并不是讲特殊的问题、工具、编程语言或特殊的机器。相反，本书讲的都是通用的技术，它们可以帮助你找到任何问题，无论你使用的是什么机器、语言

① 尖发经理（pointy-haired manager），指发型向上翘起，这是斯科特・亚当斯（Scott Adams）绘制的漫画Dilbert中的一个非常有趣的角色，他管理一家高科技公司的一个部门，但看上去对他下属所做的事情却毫不知情。

和工具。本书讲了一种查找问题的全新方法，例如，它不是告诉你如何在Glitch-O-Matic数字逻辑分析器上设置触发器，而是告诉你为什么必须使用分析器，即使把它挂接到系统需要费很大一番工夫。

本书也适用于修复各类问题。你的系统可能在设计、构建和使用中有错误，或者只是被破坏了，无论是什么情况，这些技术都将帮助你快速找到问题的核心。

本书介绍的方法甚至不仅限于工程领域，虽然它们都是从工程环境中总结出来的。这些方法也可以帮助你查找其他方面的问题，例如汽车、房屋、音响设备、管道，甚至是你的身体（本书中会有例子）。但不可否认的是，有些系统并不适合使用这些技术，例如经济学就不适用，因为它太复杂了。

1.5 本书的主旨不在预防、保证或筛选

虽然本书介绍的方法和系统都是通用的，但它们都紧紧围绕一个重点，那就是查找bug的根源并修复。

本书所讲的并不是像ISO-9000、代码评审或风险管理这样的质量改进过程，这些过程主要强调的都是防止bug的产生。如果你对这方面的内容感兴趣，我可以推荐几本好书，例如*The Tempura Method of Totalitarian Quality Management Processes*和*The Feng Shui Guide to Vermin-Free Homes*。质量保证过程所涉及的技术都很有价值，但它们往往不易实现，即使实现了，系统中仍然会留有一些bug。

一旦有了bug，就必须要检测它们，这项任务一般由质量保证（QA）部门来完成，如果没有这个部门，那么就只能由客户方来做了。本书也不会讨论这个阶段，因为已经有很多资源详细讨论了测试覆盖分析、测试自动化和其他质量保证技术。当你在产品线上检查6 467 826种选项组合时，可以找本这方面的经典书来打发时间，例如*How Do I Test Thee, Let Me Count the Ways*。

迟早会有一种组合失败，这时某位质量保证人员或客户就会起草一份bug报告。接下来，一些经理、工程师、销售人员和客户支持人员可能会召开一次“bug筛选会议（triage

meeting)”，激烈地讨论这个bug的重要性，以及是否需要修复它（何时修复）。虽然这个主题与你的市场、产品和资源密切相关，但本书绝对不会去触及它。但是，当人们决定修复bug时，肯定会看一下bug报告并且想弄清楚 “这究竟是怎么发生的”，这时就到了阅读本书的时候了（参见图1-1）。

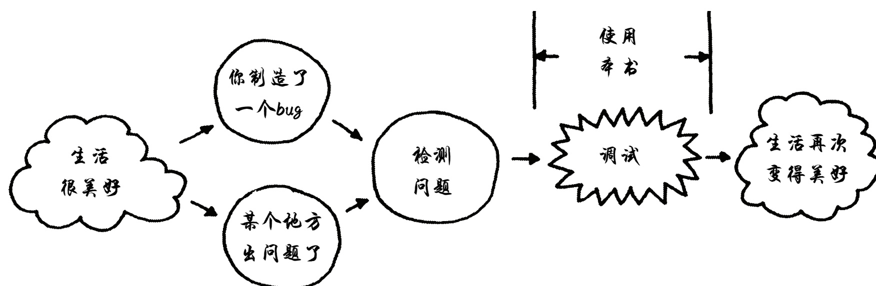


图1-1 何时使用本书

下面的章节将教给你如何准备查找bug，如何挖掘并仔细审查各种线索，以便找到根源，追踪实际问题，并修复它，然后确认你已经修复问题，这样你就可以高高兴兴地回家了。

1.6 调试不仅仅是故障检修

虽然调试和故障检修（troubleshooting）这两个词常常混用，但它们实际上还是有区别的，而且本书作为一本调试书，与其他数以百计的故障检修指南也是存在区别的。调试通常是查明为什么一个设计没有按计划工作，而故障检修通常是在已知设计没有问题的情况下，查明一件产品出了什么问题——可能是某个文件被删除了，某条线断了或者是某个元件出了问题。软件工程师做调试工作，汽车机修工做故障检修工作，而汽车设计师做调试工作（在理想世界中）。医生给病人看病就相当于“故障检修”，医生永远没有调试的机会了。（因为上帝已经完成了调试的任务，他花了一天时间设计了人类并造出原型，又在同一天把他的产品投放到人间，看起来上帝好像很赶时间！我想我们可以原谅他优先处理重要的方面，而给我们留下了像“拇指外翻[①]”和“男性规律性脱发”这样的小bug。）

① 医学术语，拇指由于囊肿胀而外翻，也称为拇囊炎。

本书中所讲的技术既适用于调试，也适用于故障检修。这些技术并不关心问题是怎么产生的，而是告诉你如何找到它。因此，无论是设计出了问题，还是部件有了毛病，都可以利用这些技术来查找。相反，有关故障检修的书只是在部件有问题的情况才有用。它们用几十张表格列出某个系统可能出现的一切症状、问题和修复方法。这些都很有用，它们是该类型的系统过去曾经出现过的一切问题的汇总，并且说明了症状和修复方法。它们把很多人积累的经验提供给故障检修人员，并帮助快速找到已知的问题。但是，当出现了新的、未知的问题时，它们就没有多大作用了。因此，这些书对设计问题几乎没什么作用，因为工程师们都非常有创造力，他们“喜欢”制造新的bug，而不会再用那些旧的bug来考验你。

因此，如果你正在检修一个标准系统的故障，那么不要忽略了规则8，查询一下故障检修指南，看看其中是否已经列出了你的问题。但如果它没有列出来，或者给出的修复方法不起作用，又或者你正在调试世界上第一个数字化的传输系统，因此根本没有故障检修指南，你不必担心，因为本书中所讲的规则将帮你找到新问题的核心。

1.7　有关案例故事

我出生于1954年，是一名美国电子工程师。在问题的讨论中，我所讲述的“案例故事”都是真实的，这些故事是我那个时代的人所熟知的。有些故事可能是你不了解的，因此有些我提到的事情你可能不理解。如果你是一位汽车机修师，可能不知道中断（interrupt）是什么。如果你生于1985年，你可能不知道什么是电唱机。但这没关系，重要的是我要通过这些故事说明的原则，而且我在讨论的过程中会给出足够多的解释，确保让你能够理解这些原则。

你知道，有些细节被我改动了，以便保护个人隐私，特别是保护那些犯了错的人。

1.8　精彩内容，即将上演

本书将介绍9条调试的黄金规则，每章介绍一条。每章的开头将讲述一个案例故事，通过它来说明规则对成功的重要性。然后描述规则并证明它如何应用于前面的故事。我会讨论

思考和使用规则的各种方式，你在面对复杂的技术问题时能很容易就想起它们（当然，简单问题也同样如此）。我还会给出规则的一些变化形式，证明它们也适用于其他方面，例如汽车和房屋。

在最后几章中，我提供了一组案例故事，用来检验你对本书的理解，还有一节是练习在一个试验性的帮助台环境中使用这些规则，最后给出一些有助于把学到的东西应用于实际工作的小技巧。

读完本书后，你的调试效率将会大大提高。你甚至会喜欢到处转转，看看哪些工程师陷入困境，然后施以援手，帮助他们化解问题。但我给你提个小建议：紧身衣和披风还是不要穿了，把它们留在家里吧[①]。

① 作者开的一个玩笑，意思是不必打扮成“蜘蛛侠”的样子，蜘蛛侠是美国电影《蜘蛛侠》里的角色，热衷于拯救别人于危难，每次出行都要穿上紧身衣和披风。

第 2 章

总 体 规 则



“我在这里要讲的理论（可能你认为它们非常荒谬），实际上都是非常实用的，我就是靠着它们挣得我这份面包和奶酪的。”

——福尔摩斯，《血字的研究》

下面就是本书要讲的规则。记住它们，把它们贴到你房子里的所有墙上。（你可能立即会想到：“调试规则”墙纸，用它们来装饰出一间别具匠心的家庭办公室，但“风水先生”肯定不会推荐你这样做。）

调试规则

规则1　理解系统

规则2　制造失败

规则3　不要想，而要看

规则4　分而治之

规则5　一次只改一个地方

规则6　保持审计跟踪

规则7　检查插头

规则8　获得全新观点

规则9　如果你不修复bug，它将依然存在

第3章

理 解 系 统

"人们要想掌握本书中所有有用知识也并非完全不可能，事实上我就是尽全力这样做的。"

——福尔摩斯，《杀人的五个橘核》



案例故事　我刚出大学校门那会儿，急于积累点经验（当然还有别的目的），于是找了份夜间的兼职工作，构建一个用微处理器操控的进料阀控制器（valve controller）。这台设备的用途是控制添加到模具中的金属粉末的进料量，它通过一台天平来称重（参见图3-1）。像很多工程师一样（特别是那些"乳臭未干"的工程师），我复制了一个基本设计（从我大学联谊会的一位好友那里弄来的，他的毕业论文使用的就是这种处理器芯片）。

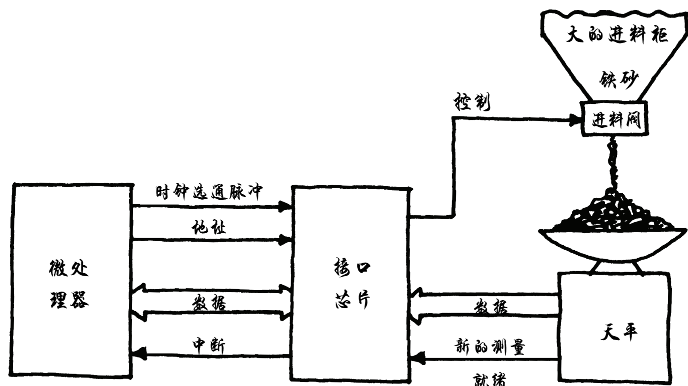


图3-1　由微处理器操控的进料阀控制器

但我的设计却无法工作。当天平为了进行一次新的测量而试图中断处理器的时候，处理器不响应它的中断请求。由于我是在晚间工作，没有很多工具可用来调试，因此费了好长时间才查明原因——芯片接收到来自天平的中断信号后，并没有把信号传递给处理器。

我当时与一位软件工程师合作，在凌晨1点的时候，我们都非常疲倦了，他坚持让我从头至尾读一遍数据手册。我照做了，在第37页，我读到“芯片将在第一个取消选中的时钟选通脉冲上中断处理器”，这等于是说“当电话铃声第一次响起时，中断就会发生，但这通电话不是打给芯片的”。然而，在我的设计中永远也不会发生中断，原因在于，为了节省硬件，我把时钟和地址线合并到一起了（我那位朋友也是这样做的）。继续电话这个比喻，只有当电话打给芯片时，电话铃才会响起。我那位朋友的系统不需要任何中断，因此可以正常工作。当我增加了额外的硬件后，我的系统也正常运转了。

后来，我同我父亲谈起了这次马拉松式的调试经过（他懂一点电子知识，但对微处理器一无所知）。他说：“这只是常识——当所有方法都不管用时，读读指令。”他的话给了我第一个启示，使我隐隐感觉到有一些通用的调试规则可以应用于计算机和软件之外的其他更多事情。（我还认识到，虽然我受过大学教育，也不足以让我父亲对我刮目相看。）这里，我要讲的规则是“理解系统”。■

在理解芯片的工作原理之前，我根本无法对问题进行调试。一旦理解了它，问题也就显而易见了。记住，系统的主要部分我都理解，这很重要。我知道设计的工作原理，我知道天平必须中断处理器的运行，也知道时钟选通脉冲和地址线都是做什么的。但我没有仔细研究所有细节，因此遇到了麻烦。

你必须掌握系统的工作原理以及它是如何设计的。在某些情况下，还要知道为什么这样设计。如果你没有理解系统中的某个部分，那么这通常就是出问题的地方。（这不仅仅是“墨

非定律”[1]的问题，如果你不能理解你所设计的系统，你的工作可能会变得一团糟。）

顺便说一下，理解系统并不等于理解问题，这有点像Steve Martin所讲的那个变成百万富翁的愚蠢方法：“首先，得到100万……”（此处的引用已得到Steve Martin的许可。）当然，你现在还不理解问题，但如果你想要查明系统为什么不工作的话，必须先理解它的工作原理。

3.1 阅读手册

理解系统的基本方法就是阅读手册。我父亲的观点也不尽然，我们应该首先阅读手册，而不是等到所有办法都不管用之后才去读它。当你买来一件东西时，手册告诉你怎么操作它，以及它是用来干什么的。我们需要一页一页读完手册并理解它，以便用它来完成我们需要做的工作。有时你会发现它不能做你需要的工作，因为你买错东西了。因此，我刚才的观点又被推翻了——在买回一块没用的废物之前，先阅读手册。

如果你的除草机不能发动，读手册可能会提醒你在拉紧除草绳之前先按几下“primer bulb”按钮。我们家有一台除草机，它开动的时候会很热，以至于会把除草绳烧化，这时就无法再工作了。而这是因为使用除草机的小伙子没有阅读手册中给除草头上润滑油的那部分内容。我读了手册之后发现了这一点。如果你做的砂锅豆腐很难吃，就要重新看一遍菜谱了。（实际上，在这种情况下看菜谱可能也不会有帮助，你最好读一下“Chu-Quik Chou House”[2]的外卖菜单。）

如果你是一位工程师，正在调试自己公司的产品，那么你需要读一读内部手册。工程师们设计它是用来做什么的？读一下功能说明以及所有的设计规范，研究一下图表、时序图和状态机。分析它们的代码，还要读一下注释。（是的，读一下注释，这非常重要。）一定要检查产品的设计。查明构建它的工程师们打算用它来做什么（除了用它来赚钱买辆宝马车以外）。

① 墨菲定律（Murphy’s Law），事情如果有变坏的可能，不管这种可能性有多小，它总会发生。

② 一家中餐馆的名字。

注意，手册上的信息也不可全信。手册（以及那些只想着赚钱买宝马车的工程师们）可能也是错的，很多难以发现的bug就出现在这里。但你仍需要了解他们的想法，哪怕其中有些信息是很难接受的。

有时，这些信息正是你需要看的：



案例故事 我们正在调试一个用汇编语言编写的嵌入式固件程序。这意味着我们必须直接分析微处理器的寄存器。我们发现寄存器B被破坏了，进一步缩小范围后发现问题是由于调用了一个子例程引起的。当我们查看该子例程的源代码时，发现在代码开头有以下注释：/* Caution—this subroutine clobbers the B register */（注意，这个子例程会破坏寄存器B）。我们修改了这个子例程，把B寄存器独立出来，于是bug被修复了。（当然，编写这段代码的程序员直接修复这个问题比输入这句注释还容易，但不管怎样他至少记录了这个问题。）

在另一家公司，我们检查过一个看起来是由于顺序错误导致的问题。我们查看了源代码，这些代码是我先前编写的，我告诉同事说我曾担心过会出这样的问题。于是，我们在代码中搜索"bug"，发现在两个函数上面有这样一条注释：/* DJA—Bug here? Maybe should call these in reverse order? */（DJA——这里是否有bug？或许应该把它们的调用顺序颠倒一下）。事实上，改变调用顺序后确实修复了bug。■

理解了你自己的系统后，还会获得一个额外的好处。当你找到bug时，必须在不破坏其他地方的前提下修复它们。理解系统行为是不破坏系统的第一步。

3.2 逐字逐句阅读整个手册

人们在调试的时候，通常都不会彻底地阅读系统手册。他们采取跳读的方式，查看他们

认为重要的一些章节，但问题的线索可能就隐藏在被略过的那些章节中。是的，我就是凌晨1点的时候找到了线索，最后终于发现了进料阀控制器的bug。

编程指南和API可能非常厚，但你必须深入挖掘它，查找你认为有问题的函数。图表部分可以忽略，它们会干扰你。但数据表要仔细查看，可能表中不起眼儿的一行指定了一个模糊的时序参数，而它就是问题所在。



案例故事 我们构建了几个版本的通信板（communications board），有些板上安装了3个电话电路，有些板上则安装了4个。3个电路的系统在现场用了一段时间后，效果良好，这时我们在实验场地引入了有4个电路的系统。这些系统所做的内部测试较少，因为两种系统所使用的电路是相同的，只是多安装了一个电路而已。

但是，在实验场地上，4个电路的系统在高温下发生了故障。我们很快在实验室中再现了故障，并发现主处理器崩溃了。这通常是因为程序内存被破坏，于是我们运行测试，结果发现内存在读回数据时发生错误。令我们感到奇怪的是，为什么安装了同样的3个电路的板子却没有出现这个问题。

硬件设计师查看了几块通信板，注意到4个电路的通信板使用了另一种不同牌子的内存芯片（它们的生产批号不同）。他查看了规格说明。两种芯片都符合工程标准，速度也都很快，而且它们读写的时序也相同。这位设计者考虑到这些规格是正确的，只是他是从处理器时序的角度来考虑的。

我把整个数据表读了一遍。我发现发生故障的内存有一个不同的规格，它指定了两次访问之间需要等待的时间。这段时间很短，看起来也不怎么重要，而且两种内存的等待时间的差别也不大，但处理器时序设计没有考虑它，而且也不满足这两种芯片的规格。因此，它在速度较慢的芯片上会频繁发生故障，而在较快的芯片上发生故障也是迟早的事情。

> 我们通过稍微减慢处理器的速度解决了这个问题，而且修订后的设计使用了更快的内存，我们还仔细检查了数据表的每一行。■

应用说明和实现指南提供了丰富的信息，它们不仅描述了系统是如何工作的，而且专门给出了先前已发生过的问题。常见错误的警告具有难以置信的价值（即使你犯的错误都很不常见）。从供应商的站点获取最新的文档，并阅读网站上所列出的最近一星期发现的常见错误。

参考设计和样本程序给出了产品的一种使用方式，有时这些就是能获得的全部文档了。但是，在使用这些设计时一定要注意，创建它们的人往往只了解他们的产品，而没有遵循好的设计实践，或者不是为真实应用而设计的（最常见的缺点是不能进行错误恢复）。不要照搬这些设计，如果你没有在开始的时候发现bug，那么将来也会发现。此外，即使是最好的参考设计可能也不会完全符合应用程序的特定需求，而不符合的地方可能就是出问题的地方。当我照搬了朋友的微处理器设计时，就发生了问题，因为他的设计无法处理中断。

3.3 知道什么是正常的

当你检查系统时，必须知道系统的正常工作状态。如果你不知道低位字节首先由使用了Intel芯片的PC程序来处理，那么你会认为所有长字（longword）都是随意处理的。如果你不知道缓存是干什么的，就会非常奇怪有些数据为什么没有马上写入内存。如果你不了解三态（tri-state）数据总线的工作原理，你将会认为它们可能是主板上的故障信号。如果你从未听说过电锯，你可能会认为那个发出讨厌的嗡嗡声的东西一定是出了什么毛病。知道什么是正常的可以帮助你注意到什么是不正常的。

你必须掌握一些你所工作的技术领域的基础知识。如果我不知道时钟选通脉冲和地址线是做什么的，那么即使我读了手册之后也无法理解中断问题。本书中几乎所有的（即使不是全部的话）示例都假定人们已经掌握了系统工具原理的一些基本知识。（如果我在前面使你误认为读完本书就可以调试任何技术领域的bug，那么请恕我无心之过。如果你是一位游戏

编程人员，最好不要去管核电厂的调试。如果你不是医生，那么就不要试图诊断你手臂上的灰绿色斑点是什么。如果你是一位政客，那么就不要介入任何有关bug的事情。）



案例故事 与我共事的一位软件工程师在一次调试策略会议结束后，边摇头边走出来。他们讨论的bug是微处理器的崩溃问题。那只能勉强算是一个微处理器，因为它没有操作系统，没有虚拟内存，也没有任何其他的东西，他们知道它崩溃的唯一线索就是它无法重新设置一个监视定时器，因此导致定时器最终超时。软件工程师们正在试图查明它在哪里发生了崩溃。有一位硬件工程师建议他们在崩溃之前设置一个断点，当到达该断点时，查看一下发生了什么情况。显然，他并没有真正理解起因和结果，如果知道在哪里设置断点，那么就已经找到问题了。

这就是软件人员和硬件人员在试图调试对方的工作时总会惹恼对方的原因。■

缺乏基础知识解释了为什么很多人找不到自己家用电脑的毛病：他们只是没有理解计算机的基本原理。如果你无法学习那些需要掌握的知识，可以遵照调试规则8，向有专业知识或经验的人请教。十几岁的孩子过马路是没问题的，但你要想让他帮你处理录像机上总是闪烁不停的“12:00”，还是等他大一些再说吧。

3.4 知道工作流程

当你尝试寻找bug时，必须知道要查找的路线。开始时，你需要猜测在哪里把系统分隔开，以便隔离问题，这种猜测完全取决于你对系统功能划分的了解。你至少要大体上知道所有的模块和接口都是做什么的。如果你的烤箱把面包烤焦了，你需要知道哪个黑色旋钮是用来控制烤制时间的。

你应该知道系统中的所有API和通信接口都是用来交换什么数据的。还应该知道每个模块或程序如何处理它们通过这些接口收发的数据。如果代码是高度模块化或面向对象的，那

么接口将很简单，模块也有良好的定义。观察接口就很容易解释你看到的东西是否正确。

当系统有一些部分是“黑盒子”时，这意味着你不知道它内部有什么，但应该知道它们如何与其他部分交互，这至少可以帮助判断问题是在内部还是外部。如果问题发生在黑盒子内部，你必须更换盒子，但如果问题出在外部，就可以修复它了。在面包烤焦的例子中，你可以控制黑色旋钮，试着把它调小一些。如果烤制时间并未缩短，说明烤箱内部出问题了，那么就扔掉它，再买个新的（也可以拆开修理一下，修不好再换个新的）。

假如你在开车时听到“嗒嗒嗒”的声音，你开得越快，声音也越急。这可能是由于轮胎面上嵌了块小石头（很好修理），也可能是由于发动机出了问题（很难修理）。当汽车高速行驶时，发动机和轮胎是同步加速和减速的。但如果你知道发动机是通过传动轴与轮胎连接的，就应知道，如果调低档位，那么在保持轮胎转速不变的情况下发动机将转得更快。于是你可以调低档位，如果声音的频率保持不变，可以推断问题出在轮胎上，于是你在路边停车，发现轮胎面上嵌了块小石头。你只需调低档位来检查传动轴，而节省了去修理店的高昂费用。

3.5 了解你的工具

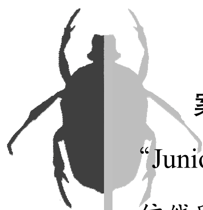
调试工具是用来观察系统的眼和耳，你必须选择正确的工具，正确地使用工具，并正确地解释得到的结果。（如果把温度计不正确的一头放到你的舌下，是不会测出正确体温的。）很多工具提供了非常强大的功能，但只有精通它们的用户才了解。你越是精通工具，就越容易查明系统中发生了什么事情。要花时间学习与工具有关的一切，通常，查明系统行为的关键（参见规则3）是你的调试器设置得怎样，或者是否正确地触发了分析器。

我们还必须了解工具的局限性。走查源代码可以显示逻辑错误，但无法显示时序和多线程问题；剖析工具可以暴露出时序问题，但显示不出逻辑错误。模拟示波器（analog scope）可以看到噪声，但无法存储太多数据；数字逻辑分析器可以捕获大量的数据，但看不到噪声。普通的体温计无法告诉你太妃糖是不是太热了，而糖果温度计的精确度甚至不足以测出你是否发烧。

那位建议在崩溃之前设置一个断点的硬件人员并不知道断点的局限性（或者他有某种奇妙的时间旅行技术）。软件人员最后采取的方法是在系统上接入一个逻辑分析器，用来记录微处理器的地址和数据总线的变化痕迹，同时把监视定时器设置成一个非常短的时间。这样，当定时器超时的时候，地址和数据总线的变化痕迹也就保存下来了。他知道必须把发生的事情记录下来，因为直到定时器超时后，他才能知道处理器已崩溃。把定时器缩短的原因是他知道分析器无法记录太多的信息，达到它的记忆量后，旧信息就会被替换掉。通过查看跟踪记录，可以看到程序在什么地方发生了问题。

你还必须了解开发工具。这当然包括用来编写软件的语言，如果你不知道C语言中的"＋＝"操作符是做什么的，代码的某个地方就会出问题。但除此之外，你还需要了解一些更微妙的知识：编译器和链接器在把代码发给机器之前会进行什么处理。数据是如何匹配的，引用是如何处理的，以及内存是如何分配的，这都将对你的程序产生影响，而这些通过源程序并不能明显看出来。硬件工程师必须知道如何按照高级芯片设计语言中的定义来设计芯片上的寄存器和门。

3.6　查阅手册



案例故事　在一家大型计算机制造企业工作的一位初级工程师（我们称他"Junior"）正在设计一个系统，系统中使用一种1489A芯片。这种芯片接收来自通信线路的信号，类似于计算机与调制解调器之间的连接。一位高级工程师（我们称他"Kneejerk"）看到他的设计后指出："哦，你不应该使用1489A，而应该使用原来的1489。"Junior问为什么，Kneejerk回答说："因为1489A会变得过热。"像所有初出茅庐的年轻人一样，Junior怀疑所有长者的话，于是他决定了解一下电路，看看新版本的芯片为什么会发热。他发现这两种芯片之间的唯一区别是一个内部偏压电阻的值不同，新的芯片在电路中具有更强的抗噪声干扰性。现在，1489A中

> 的这个电阻较小，因此，如果对它施加过大的电压，它就会发热。但是在目前这个设计中，电阻的连接方式并不会对它施加过大的电压，当然也就不足以令它过热。理解了这个电路后，Junior没有采纳Kneejerk的建议，仍旧使用了1489A。事实也证明它没有过热。
>
> 几个月后，Junior开始检查小组先前设计的电路。当他按照原来的1489的图示把示波器探针放在输入引脚上时，引脚的读数看起来是错误的。他查了查已经被他翻得很旧的数据手册，确信是引脚线接错了——他们把输入接到了偏压电阻上，而没有接到输入引脚上。偏压电阻的引脚通常是不接线的，但如果把输入接到它的引脚上，它也能工作，但这样电路就完全失去了抗噪声干扰的能力。这个错误连接还使得电流绕过了输入电阻，导致大量电流流经内部偏压电阻。事实上，Junior注意到了一个有趣的现象——元件会发热，而且如果使用1489A的话发热量会更大。■

在这个故事中，有两个地方违反了“理解系统”规则。首先，原来的工程师在设计电路时没有查阅引脚的编号以确保连接正确。随后，Kneejerk使问题进一步复杂化，他没有通过理解电路来查明为什么新元件会发热。结果，这个团队所设计的电路使用了一种已经过时且很难找到的元件，这种元件散发大量热量而又丝毫没有抗噪声干扰的能力。除此之外，其他的工作完成得都很出色。

相反，Junior没有相信示意图上的引脚连线，他查阅了数据手册中的正确连接方法，因此知道了元件为什么会发热。

不要猜测，而要查阅手册。芯片制造商或软件工具的开发人员已经把详细信息写到手册中，而你不应该盲目相信自己的记忆。养成良好的查阅习惯，无论是芯片的引脚连接，还是函数的参数，甚至是函数名称。我们要学爱因斯坦，他从不记忆自己的电话号码，“干嘛要费事记它呢？它不就在电话簿中吗？”

如果你单凭猜测去观察芯片上的信号，那么当你看到错误的信号时，可能会把它当成正确的。如果你假定函数的参数调用顺序是正确的，那么可能会像原来的设计者那样把问题忽略过去了。这会导致信息的混淆，甚至再一次确认了错误的信息。不要把调试的时间浪费在那些错误的信息上。

最后提醒你一点，如果在深夜2点你家的地下室因为水管坏了而被淹没，当你修不好而决定打电话求助时，不要乱猜电话号码，去查电话簿吧。

3.7　小结

理解系统

这是第一条规则，因为它是最重要的。

- **阅读手册**。它会告诉你在使用除草机时，要在除草头上涂润滑油，这样除草绳就不会被烧化。
- **仔细阅读每个细节**。有关微处理器如何处理中断的详细信息就隐藏在数据手册的第37页。
- **掌握基础知识**。电锯本来就会发出很大的噪声。
- **了解工作流程**。引擎的转速可能与轮胎的转速不同，这是由传动轴造成的。
- **了解工具**。弄清楚体温计的哪一端才是用来测量体温的，弄清楚Glitch-O-Matic逻辑分析器的强大功能是如何使用的。
- **查阅细节**。连爱因斯坦都会去查阅细节，而Kneejerk却盲目相信自己的记忆力。

第4章

制 造 失 败

“什么也比不上直接取得的证据来得重要。”

——福尔摩斯，《血字的研究》



案例故事　1975年的一天深夜，我独自一人在实验室里调试一款电视网球游戏的一个问题。这是第一批家庭电视游戏当中的一个，它是由当地的一位企业家出资，在MIT创新研发中心开发出来的。游戏中有一面用来练习击球的墙，我要解决的bug就发生在球从墙上反弹回来的那个时刻，但它只是偶尔才发生。我把示波器调好（这台示波器有点像老的科幻电影中的那些机器，它在一个小的圆形屏幕上画出波形曲线），准备观察bug，但发现我很难在示波器上观察到bug。因为如果我把球速调得很慢，它隔好几秒钟才能从墙上弹回一次，但如果把球速调得太快，又不得不把所有注意力都放在击球上。如果漏球了，就得等到下一次发球再去观察。这并不是一种高效的调试方式，我不由想到：“算了，这只是一个游戏而已。”但我很快打消了这个念头，我想如果能让游戏自己玩起来，那我就能集中注意力观察示波器了。

我发现击球板的上下位置和球在各个方向（上下左右）的位置都是用电压表示的（参见图4-1）。（如果你对硬件知识不够了解，那么我来解释一下什么是电压。电压就像是一个容器中的水平面，我们通过注水或排水来改变水面的高度。电压与这类似，只是它没有水，而是电荷。）

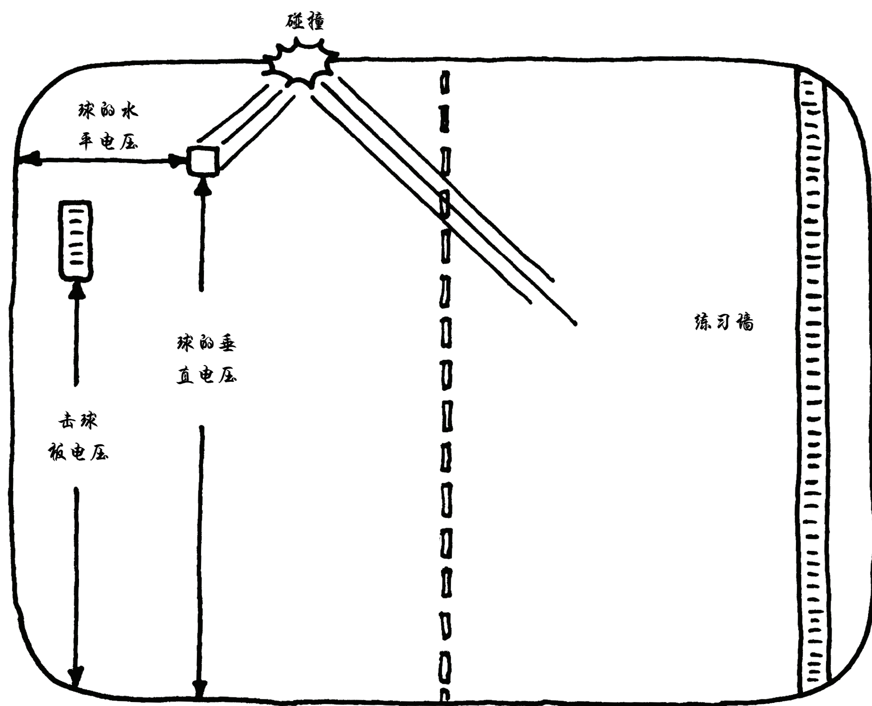


图4-1　电视网球游戏

我意识到，如果把击球板的电压连接到控制球上下位置的电压，而不是连接到手动操纵杆，那么击球板就能跟随球一起移动。这样，当球被弹回来时，击球板就能找准高度把球击回去。我按照这种思路把电路连好，击球板打得非常好！游戏终于可以自己玩起来了，我可以集中精力观察示波器，很快就发现并解决了问题。■

即使是在深夜，我也能很容易在示波器上看到球在反弹回来时是如何发生错误的。关键是在发生失败的时候要看到它。这是很多调试的典型问题——你看不到问题是如何发生的，因为你方便观察的时候，它并没有发生。这并不是说它只发生在深夜（即使你最终是在深夜时发现了它），它可能是每7次中只发生1次，或者是Charlie测试它的时候，碰巧发生了一次。

如果Charlie现在正在我的公司工作，你问他："当你发现一个故障时该怎么办？"他会

回答说："试着让它再次发生。"（Charlie是一位训练有素的调试人员）。这样做有3个原因。

- **可以观察它**。要观察错误（下一节将更详细地讨论这个问题），就必须使它发生。我们必须尽可能有规律地制造失败。在前面讲的电视游戏的例子中，当问题发生时我可以集中注意力观察示波器（虽然当时我已经很疲倦）。
- **可以专心查找原因**。准确地知道问题在什么条件下会发生，有助于集中精力查找原因（但是请注意，有时这会产生误导，例如，"烤箱只有在你把面包放进去的时候才会把面包烤焦，因此问题就出在面包上。"这个问题后文也会详细讨论。）
- **可以判断是否已修复问题**。当你认为已经修复了问题时，如何才能确信它确实已被修复呢？那就是明确知道问题是如何发生的。当问题没有修复时，如果你执行X操作，失败率为100%；在修复问题后，再执行X操作，如果失败率为0，那么你知道bug确实已被修复。（我这么说并不多余。很多时候，开发人员在修复bug时会修改软件，然后在一个与当初发现bug的不同条件下测试新软件。软件当然能运行，即使他在代码中输入一行打油诗，而他也高兴地回家了。然而，几星期后，在测试过程中，或者更糟，在客户现场，软件再次失败。后文将讨论更多这方面的内容。）



案例故事　我最近买了一辆四轮驱动的新车，整个夏天我开着它都没发现什么问题。当天气开始变凉时（在新罕布什尔州，一般到9月天气才开始凉爽），我注意到，如果时速变为25至30英里/小时之间，头几分钟内，车子后部会发出"嗞嗞"的噪声。加速或减速，噪声会消失。10分钟后，噪声也会消失。如果温度高于25℉[①]，也不会发出噪声。

我把车开到经销商那里进行常规保养，我告诉他们在早上天气较凉的时候先把我的车子开出去，听听声音。他们却没有照我说的去做，直到上午11点才开始看我的车，这时温度已经达到37℉了，车当然没有发出噪声。他们卸下轮胎，检

① 华氏温度T_1与摄氏温度T_2的换算关系为 $T_2 = \frac{5}{9}(T_1 - 32)$。——编者注

查了刹车，没有发现问题（当然不会发现）。他们没有制造失败，因此无法找到问题。（我一直想等一个较冷的天气再次把车开到那里，但我们遇到了历史上最暖的一个冬天。这就是墨菲定律。我想在车子过保修期之前，总会有一个冷天气吧。）■

4.1　制造失败

但如何才能让它失败呢？一种简单的方法是进行一次内部预演，还有一种同样有效的方法是演示给未来的投资者。如果碰巧没有客户或投资者在现场，那么你就必须设法正常使用，并观察它是如何出错的。当然，测试本来就应该是这样的，但这里的重要之处是在错误第一次出现之后，能够使它再现。通常，认真记录测试过程可以作为补充，但你必须认识到仅有一次错误是不够的。当一个3岁的孩子看到她的父亲从梯子上摔下来，失手打翻了油漆桶，弄得满头满脸都是油漆时，她会拍手大叫："再来一次！"我们应该向这个孩子学习。

仔细观察你做了什么，然后再做一次，并且记下你做的每个步骤。然后，按照你自己所写的步骤去做，确定这样做确实导致了错误。（把油漆洒到头上的父亲就不要再这样做了。实际上，在有些情况下，令设备发生错误具有一定的破坏性，或者代价很大，这时每次都使设备发生同样的错误就不是一种好办法。为了控制损失，必须改变一些地方，但我们应该尽量少改动原来的系统和顺序。）

4.2　从头开始

通常，所需的步骤很短，也很少。例如，单击这个图标，就会出现拼写错误的消息。有时，虽然步骤很简单，但需要进行很多设置。例如，重启计算机，运行这5个程序，然后单击这个图标，出现了拼写错误的消息。由于bug可能仅仅在机器的某个复杂状态下才会出现，因此必须仔细注意机器在执行这些步骤时的状态。（如果你告诉修车工每当在寒冷的天气里开车时，车窗就会被冻住而打不开，他应该会猜出你每天早上都洗车。）试着从一个已知的状态开始，例如刚刚重启的计算机，或者是你一早步入车库时汽车的状态。

4.3　引发失败

在调试故障的时候，如果需要手工执行很多步骤，那么使这个过程自动化会很有帮助。前面的电视游戏就是这种情况，我需要同时玩游戏和调试，自动的击球板替我完成了玩游戏的任务。（很遗憾我不能把调试工作自动化，而由我来玩游戏。）在很多情况下，只有在重复很多次后，错误才会出现，因此我们希望在夜间运行自动测试工具。软件很愿意整夜工作，你连比萨饼都不用为它买一块。



案例故事　我房屋的一扇窗户漏雨，但只是在倾盆大雨而且刮东南风的时候才会漏。我想赶在下一次暴风雨到来之前把它修好，于是我架了把梯子，拿着喷水的水管向窗户喷水，看看它到底是怎么漏雨的。这使我准确地看到了漏雨的地方，原来是墙缝那里有一道空隙，我把这个空隙塞满，并再次喷水检查，发现即使在这么高的水压下，它也不再漏水了。■

过敏症专科医师会用各种已知状态的过敏原对病人皮肤进行实验，看看哪些会引起反应。牙医会在病人口腔中喷冷空气，以便发现对冷物敏感的牙齿。（另外，牙医这么做也会觉得很有趣。）州警在检查醉酒驾车时会让司机走直线，身体向后仰，让司机用手摸自己的鼻子，以及倒背字母表等事情，以确定他是否酒后驾车。（这比让司机开一段车，看看他是否能够在高速公路的正确一侧行驶要安全得多。）

如果你的Web浏览器有时会打开错误的网页，就应该通过设置浏览器来让它自动请求页面，以便查找错误。如果网络软件在高流量下发生错误，应该运行一个网络加载工具来模拟负载，这样就可以引发错误。

4.4　不要模拟失败

引发失败（正确）和模拟失败（错误）这二者之间存在着非常大的差别。在前面的例子

中，我建议模拟网络负载，而不是模拟失败机理本身。正确的方法是模拟那些导致失败发生的条件。但是，不要试图模拟失败机理本身。

你可能会问："为什么要这么做呢？"如果有一个间歇性的bug，你可能猜测它是由某个底层机制引起的，于是构建一个配置来模拟该底层机制，以为这样就可以反复观察到bug。或者，你可能在发生bug的现场之外来模拟它，方法就是在你自己的实验室中建立一套等价的系统。以上两种方法都是在试图模拟失败，即重新制造它，只是采用了另一种不同的方式或系统。

如果你猜测失败机理，模拟往往不会成功。原因通常有两个，要么你的猜测是错误的；要么测试改变了条件，模拟的系统可以正确工作，或者更糟，发生新的错误，因而分散了你对正在查找的问题的注意力。举个例子，如果你的字处理程序（这是你要拿来与Microsoft媲美的程序）在保存文件的时候总是删除图片，你可能猜测这个问题是在写文件时发生的。于是你编写了一个测试程序，它不断地向磁盘写文件，最后操作系统死机了。这样，你得到的结论就是Windows的速度太慢了，因此你的解决办法就是开发一个用于淘汰Microsoft Windows的操作系统。

查找现有bug已经够我们忙的了，不要再制造新的bug。利用工具来观察发生了什么错误（参见规则3），但不要改变机理，因为正是这个机理导致了错误。在字处理器的例子中，不要改变文件写入磁盘的方式，而应该自动生成按键，然后观察什么被写入了磁盘。

在类似的系统上再现bug是一种较为有用的方法，但有一些限制条件。如果一个bug可以在多个系统上再现，那么我们就可以认为它是设计bug，因为它并不是在一种系统上以某种特定的方式出现。如果在某些配置下能够再现它，而在另一些配置下无法再现，那么这就帮助我们缩小了查找范围。但是，如果无法快速地再现它，那么不要为了使它出现而改变你的模拟环境。这样会产生新的配置，而不是原来发生错误的那个配置了。无论一个系统在哪种常规配置下发生故障（即使是间歇性故障），都要在该系统上使用该配置来查找问题。

一种典型的情况是在客户现场的某种综合条件下发生问题——软件在某台机器上驱动某个特定周边设备时失败。通过在你自己的现场建立一个相同的配置，或许可以模拟失败。

但如果没有相同的设备或条件，因而无法模拟失败，那么你可能会试图模拟设备或发明新的测试程序。不要这样做，而应该克服困难，要么把设备运到你自己的场地，让工程师来调试，要么派工程师（用出租车载上插装工具）到客户现场进行调试。如果客户现场位于阿鲁巴岛[①]，那么运输设备是不太现实的。顺便问一下，贵公司最近是否雇到了既擅长写作又具有丰富调试经验的好手[②]？

注意，不要用一个看似完全相同（而实际上不同）的环境来代替并希望看到相同的错误。当我修理漏雨的窗户时，如果我假设窗户的设计有问题，那么我可能会用另一扇“完全相同”的窗户来做实验。这样就不会发现墙边的缝隙了，因为这条缝隙是那扇漏雨的窗户才有的。

问：有多少工程师参与修复电灯泡？

答：一个也没有。他们都说：“我不能让失败再现，因为我办公室的灯泡很正常。”

记住，这并不意味着不能用自动化过程来引发失败，也不意味着在这个过程中不能采用一些起到放大效果的措施。自动测试能够使间歇性的问题更快发生，例如电视游戏的例子。放大效果可以使得细微的问题更明显，例如在修窗户的例子中，我可以用喷水管来找到漏雨的窗户，而不用等待偶尔才有的暴风雨来检查。这两种技术都有助于引发失败，而不是模拟失败的机理。所做的改变应该是一些高层次的改变，只影响错误发生的频率，而不影响错误的发生方式。

此外，还要注意不要画蛇添足，引发新的问题。不要因为假设芯片的问题是由于热量引起的，就用热风枪来给芯片加热以模拟错误，这样只会把芯片烧化，然后你会误认为bug完全就是电路板上那堆被烧化的塑料。如果我用消防用的水龙来检查漏雨问题，可能会断定问题显然就是出在被击碎的窗子上。

4.5　如何处理间歇性 bug

当故障只是偶尔发生时，用“制造失败”这种方法来调试就困难得多。很多棘手的问题

① 阿鲁巴岛，Aruba，安的列斯群岛中的一个岛。

② 意思是：“我就是这样的一个调试高手，不如雇我吧。”

都是间歇性的，这就是不能总是应用这条规则的原因——它很难应用。你可能已经制造出了一次失败，但是当你用同样的方式再次尝试时，问题仍然间歇性出现，可能5次、10次甚至几百次中才会出现一次。

关键问题在于你并没有完全弄清楚失败是如何发生的。你知道你做了什么，但并不知道完整的、准确的条件。还有其他你没注意到或无法控制的因素，例如初始条件、输入数据、时序、外部过程、电子噪声、温度、振动、网络流量、月相（phase of the moon）以及测试者是否清醒，等等。如果你能够控制所有这些条件，那么就可以一直使错误发生。当然，有时你无法控制这些条件，我们将在下一节讨论这个问题。

那么，如何控制这些条件呢？首先，查明它们。在软件中，查找未初始化的数据（它们总是带来麻烦）、随机数据输入、时序误差、多线程同步和外部设备（例如电话网络，或者看看是不是有6 000个孩子在同时点击你的站点）。在硬件中，查找噪声、振动、温度、时序和部件误差（类型或供应商）。在我那部四驱车的例子中，如果我没有注意到温度和车速，那么问题看起来就是间歇性的。



案例故事　一家老式的大型计算机中心间歇性地在下午发生崩溃，虽然它几乎是在同一时间崩溃，但并不总是在程序的同一个位置崩溃。人们最后发现崩溃时间与下午三点人们喝咖啡短暂休息的时间吻合。这时自助餐厅中的所有自动售货机都在同时操作，导致硬件的电力供应不足。■

一旦想到了有哪些条件可能影响你的系统，必须大量尝试与这些条件相符的各种形式。初始化这些条件，并按照一种已知模式把这些条件作为你的问题软件的输入。尝试控制时序，然后改变它，看看系统在某个特殊设置下是否会失败。对有问题的电路板进行多种测试，例如振动、加热、制冷、注入噪声以及改变时钟速度和电压，直到失败频率出现变化。

有时，你会发现当你控制某个条件的时候，问题消失了。这时你就发现了是什么（随机

产生的）条件导致了失败。当然，如果发生这种情况，你需要尝试该条件下的每个可能的值，直到找到导致系统失败的那个值。如果一个随机的数据输入模式间歇性地导致系统失败，而固定的数据模式不会导致失败，那么就要尝试每个可能的数据输入模式。

有时，你会发现有些条件是无法控制的，但可以增加它的随机性。例如，振动一块电路板或注入噪声。如果故障是由某个低概率的事件（例如噪声峰值）引起的，那么问题就是间歇性的，这时可以通过增加条件（噪声）的随机性，来提高这些事件发生的频率。这样，错误就会更频繁地发生。这可能是我们所能采取的最好办法了，它能提供很大的帮助，可以告诉我们失败是由什么条件引起的，也能使我们更容易看到失败。但有一点要注意，在对条件进行放大操作的时候，不要引起新的错误。如果一块电路板有一个对温度很敏感的错误，而你却决定振动它，以至于所有芯片都松动了，那么将会有更多错误，而这些错误与原来的错误毫无关系。

有时，你的所有尝试都不会有任何区别，你又回到起点，问题仍然间歇性地发生。

4.6　如果做了所有尝试之后问题仍然间歇性发生

记住，我们之所以制造失败，是出于3个目的：一是观察错误，二是查找线索，三是确认是否已修复。下面就讨论一下当问题看起来“有它自己的思维”时，应该如何完成这3个目标。记住，问题是没有自己的思维的，失败肯定有原因，你一定能够找到它。它只是“巧妙地”隐藏在你尚未发现的大量随机因素背后。

4.6.1　仔细观察失败

你必须能够看到失败。如果它不是每次都发生，那么就必须忽略掉不发生的时候，而在它每次发生时观察它。关键是在每次运行的时候捕捉相关信息，以便在发生失败之后查看这些数据。方法就是让系统在运行的时候尽可能多地输出信息，并把它们记录到“调试日志”文件中。

通过查看捕获到的信息，很容易把正常运行和错误运行放一起进行比较（参见规则5）。

如果你捕获到了正确的信息，就能够看到正常运行与错误运行之间的区别。仔细观察只在错误运行中才发生的那些事情。这是实际开始调试时需要注意的地方。

尽管失败是间歇性的，但这样就能识别并捕获发生错误的条件，然后对其进行分析，就像它们每一次都发生一样。



案例故事　我们的视频会议系统发生了一个间歇性的错误，在我们的部门呼叫另一家供应商的部门的时候。大约每5次呼叫中，就会有一次呼叫发生错误，导致对方的系统关闭视频，只留下音频电话呼叫。

我们无法调查对方的系统，但可以记录自己的调试日志。我们捕获了两次连续呼叫的数据，前一次是正确的呼叫，后一次则发生了错误。在失败的呼叫日志中，有一条消息显示我们发送了一条异常的命令。我们检查了前一次正常的呼叫以及其他正确的呼叫，发现所有正确呼叫中都没有这条消息。我们记录了更多日志，直到再次发生一个错误的呼叫，发现日志中又出现了那条异常命令的消息，这使我们非常确信错误就发生在这里。

后来我们查明问题出在内存缓冲器上，它里面装满了前一次呼叫的命令，而在新呼叫开始发送命令之前可能没有清空它们。如果缓冲器正确清空，则一切就会正常。如果没有清空，我们发现的那条异常命令就会在呼叫开始的时候被发送出去，对方的机器就会错误地解释它并进入只有音频的工作模式。■

我们之所以能够看到这个系统错误，就是因为跟踪了每次呼叫，获得了足够多的信息。虽然错误是间歇性的，但日志显示了它每次发生时的情况。

4.6.2　不要盲目相信统计数据

制造失败的第二个目的是获得问题发生的线索。当发生一个间歇性问题时，你可以注意

那些看起来与问题有关的操作模式。这种思路是没有问题的，但不要被表面现象所误导。

如果失败是随机发生的，你可能无法收集到足够多的统计样本来作出判断，例如，用左手点击按钮与用右手点击按钮是否有着很大的区别。在很多时候，巧合会使你误认为某种条件比其他条件更可能引发问题。然后你就会开始仔细研究“这两种条件之间有什么区别”，由于你找错了对象，这将会浪费大量时间。

这并不意味着你所看到的这些巧合的区别与问题不存在任何联系。但是，如果它们没有直接的影响，那么它们与问题的联系将会隐藏在其他随机因素背后，这时通过查看这些区别来找到原因的机会是非常渺茫的。可能你在拉斯韦加斯赢钱的机会都要比这大一些。

当你捕获到足够多的信息时（像前一节所描述的那样），就可以确定哪些因素总是与bug有关，或者哪些因素从来都与bug无关。在查找问题根源的时候，这些因素是需要重点关注的。

4.6.3 是已修复bug，还是仅仅由于运气好，它不再发生了

如果失败是随机发生的，那么要想证明bug是否已被修复就会困难得多，这一点是毫无疑问的。如果在测试的时候，每10次发生1次失败，在你“修复”它之后，变成了每30次发生1次，而你在测试28次之后终止了测试，这时你认为问题已修复，但实际上并没有。

如果采用统计测试的方法，那么运行的样本越多，结果就越准确。但是，更好的方法是找到一个总是与失败有关的事件序列。即使这个序列本身就是间歇性的，但当它发生时，100%会发生失败。然后，当你认为已修复bug时，就可以运行测试，直到这个序列出现，如果没有发生失败，那么你确实已修复了bug。这样，你在试验28次之后不会终止测试，因为你还没有看到那个序列出现。（或者，你可能在试验28次之后停止了测试，但原因并不是你认为bug已修复，而是比萨饼送到了，你在吃过晚餐之后还会回来继续测试。这也反映出了为什么应该使用自动测试的另一个原因。）



案例故事　我们的视频呼叫系统在同时使用6根电话线路时会间歇性地发生一个问题。视频会议系统在工作的时候，每秒钟要处理大量的字节，仅有一根电话线是不够的。因此系统使用6条线路来处理电话呼叫，并且把视频数据分配到这6条电话线路上。

问题在于，电话公司可能不会使用同一个路径来传送所有6个呼叫，因此某些呼叫的数据可能比其他呼叫的数据晚些到达。为了解决这个问题，我们使用一种称为“捆绑标记”（bonding）的方法，即发送端的系统在每个数据流中放置一个标记信息，这样接收端就知道了每个数据流的延迟时间。然后，接收端为传输速度快的数据流增加其自己的延迟时间，直到它们再次排列好顺序（参见图4-2）。

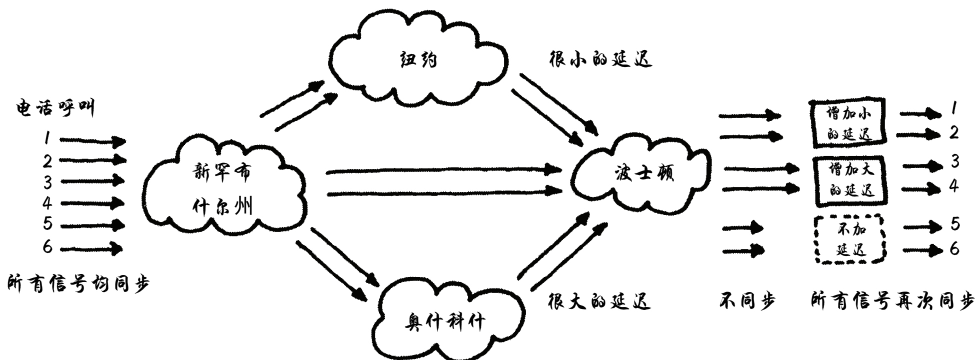


图4-2　电话网络的捆绑机制

有时，我们的系统会收到杂乱的数据，发生的频率大概是每5次发生1次。但在其他一些时候，也可能连续60次呼叫也不会发生错误。由于捆绑问题可能会引起这种杂乱的数据，因此我们在系统中安装了一个工具，用来打印与这6个呼叫有关的信息。我们发现，在大部分时间里，电话公司是按正常顺序连接这6个呼叫的，即1、2、3、4、5、6。但当发生错误的时候，连接顺序就不是正常的了，例如1、3、2、4、5、6。

> 这为我们查找问题原因提供了一条线索（主要检查那些顺序不正常的呼叫）。通过进一步检查，我们发现了bug。修复问题后，我们运行了大量测试，我们不关注呼叫顺序正常的情况，而只注意顺序不正常的情况。当发生呼叫顺序不正常的情况时，电话呼叫并没有发生错误，因此我们知道问题确实已被修复。■

这个有趣的顺序问题与电话公司的信号流量有关，而且与时间段和附近小镇上的年轻人打电话的习惯有关。在检查错误是否已被修复的时候，如果只是等待，看看故障是否还会发生，那么在等待的这段时间内如果电话流量很小的话，可能会对我们产生误导。但是，当我们把呼叫顺序与故障联系起来时，那么就可以直接测试故障是否已修复，而不必等待电话流量高峰期的出现。

4.7 “那不可能发生”

如果你曾经与工程师们打过交道（与他们一起工作过足够长的时间），那么一定听他们说过“那不可能发生”。测试人员或现场技术人员报告了一个问题，而工程师则摇摇头，思考一会儿，然后说：“那不可能发生。”

有时，工程师确实是正确的，测试人员搞混了。但更多的时候测试人员并没有搞混，问题是确实存在的。然而，在很多情况下，工程师在某种程度上也是对的——“那不可能发生”。

这里的关键是“那”这个词。“那”是指什么？它是测试人员或工程师所认为的问题背后的失败机理。或者说“那”是指一个事件序列，这个序列看起来是再现问题的关键。而且，事实上，“那”可能确实不会发生。

但是，失败的的确确发生了。我们并不清楚是什么测试序列触发了它，也不知道它是由什么bug引起的。那么，下一步就是忘掉所有假设，让它在工程师面前再次发生。这样就会证明你报告的测试序列是正确的，而且可以让工程师收回他所说的“不可能发生”这样的话，

或者尝试一种新的测试策略，指明问题的真正根源隐藏在哪里。



案例故事　“侃车”节目的两位主持人曾经给听众出过一个非常有趣的问题。故事是这样的，一位车主抱怨说他那部1976年产的沃拉雷牌（Volare）汽车在他和家人到冰淇淋店买东西时，总会发生启动问题。他们经常去当地一家冰淇淋店，要么买香草和巧克力味的冰淇淋；要么买一种特殊口味的薄荷冰淇淋，它是由3种原料混合制成的——豆腐（由三色豆子加工而成）、薄荷和牛肉片。如果他们买香草和巧克力冰淇淋，汽车就会正常发动。如果他们买后一种冰淇淋，车子在发动的时候就会发出很大噪声，最后开起来后的车况也非常糟糕。

这个问题的答案是，香草和巧克力味的冰淇淋很畅销，因此都是预先用一夸脱[①]的容器包装好的，可以随时拿给客人。但买薄荷冰淇淋的人并不多，因此需要手工包装。在炎热的夏天（虽然是傍晚），手工包装花费的时间足以使一部老的沃拉雷发动机发生汽塞（vapor lock）问题。（摘自cars.com 的Car Talk版块，此处的引用已得到许可。）■

你可能会非常肯定地认为冰淇淋的口味并不能影响汽车。你是正确的，这的确不可能发生。但买一个怪味冰淇淋却可以影响汽车，你只有接受这个数据并仔细分析具体情况，才会发现“那个”问题。

4.8　永远不要丢掉调试工具

有时，一种测试工具可以在其他的调试场合重复使用。当你设计它的时候，应该考虑到这一点，并且使它易于维护和升级。这意味着要采用好的工程技术，并实现文档化，等等。把它加入到源代码控制系统中，并构建到你的系统中，以便随时可以使用。不要只把它当做

① 1夸脱约等于1.1012209升。——编者注

一次性的工具来编码，扔掉它可能是错误的。

有时，一个工具非常有用，你实际上可以把它当做产品来卖。很多公司发现所开发的工具比产品还要畅销，于是开始转而销售这种工具。工具的用处可能是你完全想象不到的，就像下面要讲述的故事一样。



案例故事　继续前面讲过的那个电视游戏的故事。几个月之后，我已经忘记了我设计出来的自动击球板，这时，我们骄傲地向那位投资项目的企业家展示我们的原型。(是的，就是这位投资人，游戏并没有出问题，这一点令我们非常满意。) 他很喜欢游戏的玩法，但并不是很满意。他抱怨说游戏没有开启两面墙的功能。(他以前曾经看到早期的原型，当停用击球板操纵杆的时候，会出现两面墙，球在它们之间来回反弹。)

我们感到非常不解。有人会需要两面墙吗？我的意思是说，你甚少要有一个击球板才能玩游戏。他像一个真正的工程师那样说：“哦，有一个问题你们忽略了，我需要这个功能来帮助我销售这款游戏。我想在商店里展示这款游戏，就像有人在玩它一样（虽然没人在玩），好吸引人们购买。球必须来回弹起来，如果有两面墙的话，就能做到这一点。”

接下来应该做什么，我想你会猜到了，但在场的人除了我之外没有一个人知道。我极力掩饰住我的兴奋之情，平静地说：“嗯，我有个主意可能管用。让我试试看。”我平静地拿起电路板，离开房间。关好门之后，我兴奋地蹦跳着出了大厅。我用最快的速度把调试电路加了上去（为了起到更好的演示效果，我还加了一个开关），不到4分钟，我就回到了房间里，装作十分平静的样子把原型摆放在会议桌的中央，并开始用手动的击球板玩起了游戏，然后，我悄悄打开了开关。自动击球板跟随着球上下移动，比任何人玩得都好，这不仅给了那位投资者一个很大

> 的惊喜，而且我的同事们也对我刮目相看了。这是我初出茅庐的最佳表现之一。
>
> 这款游戏在上市的时候提供了练球墙和“练球板”两个选项，商店同时运行这两种选项，这使得它非常畅销。■

4.9　小结

制造失败

虽然看起来很简单，但如果不制造失败的话，调试就会变得很困难。

- **制造失败。**目的是为了观察它，找到原因，并检查是否已修复。
- **从头开始。**修车工需要知道汽车车窗在被冻结之前你洗过车。
- **引发失败。**用喷水管向漏雨的那扇窗子喷水。
- **但不要模拟失败。**用喷水管向漏雨的那扇窗子喷水，而不要向另一扇不同的、“类似的”窗子喷水。
- **查找不受你控制的条件（正是它导致了间歇性失败）。**改变能够改变的每件事情，振动、摇晃、扭曲，直到再现失败。
- **记录每件事情，并找到间歇性bug的特征。**我们的绑定系统总是只在呼叫顺序错乱时才会失败。
- **不要过于相信统计数据。**绑定问题看起来与时间段有关，但实际上真正的原因是当地的年轻人占用了电话线路。
- **要认识到“那”是可能会发生的。**甚至冰淇淋的口味也会影响汽车的发动。
- **永远不要丢掉一个调试工具。**自动击球板可能在某一天就会派上用场。

第 5 章

5

不要想，而要看

“在没有事实作为参考以前妄下结论是个很大的错误。主观臆断的人总是为了套用理论而扭曲事实，而不是用理论来解释事实。”

——福尔摩斯，《波希米亚丑闻》



案例故事 我们公司设计了一种可以插入到个人电脑中的电路板，它有自己的从属微处理器和存储器（参见图5-1）。主计算机在启用从属处理器之前，必须下载从属处理器的程序存储器（program memory）。这是通过从属处理器的一种特殊机制完成的，利用它可以把主机的数据通过从属处理器发送给存储器。数据发送完成后，从属处理器检查存储器的内容是否有错误（使用一种称为“检验和”的有效机制）。如果存储器内容正确，则从属处理器开始运行。如果内容不正确，从属处理器将报告给主机，并假定程序出错，因此不会运行。（显然，这个微处理器如果搞政治肯定会四处碰壁。）

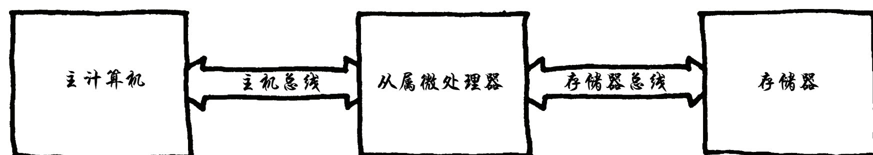


图5-1　出错的系统

问题是，在数据下载完成之后，从属处理器有时会报告一个错误，然后停止运

行。这是个间歇性的问题，10次当中可能会发生1次，而且只在一些系统中发生。当然，在下载的65 000个字节中，只要有一个坏的字节就会导致错误。公司派几位初级硬件工程师来查找并修复这个问题。

他们首先编写了一个测试程序，通过主机的总线把主机的数据写到从属处理器的一个寄存器中，然后再把数据写回主机中。他们把这个"写回"测试运行了数百万次，发现写回的数据一直是正确的。因此，他们说："好了，我们已经测试了，把数据从个人电脑的主机总线写到了从属处理器中，这个测试表明一切正常，因此问题肯定出在从属处理器与它的存储器之间。"他们查看了存储器接口（试图了解电路，这当然是对的），发现在时间的设计上有点问题——保持时间太短了，因此认为这种设计只能勉强接受。"哎呀，"他们说道，"或许存储器的地址没有足够的保持时间，哪怕它是被看做一个无粘接的接口[①]。"（初级工程师的确都是这么说话的，不过有些人会使用语气比"哎呀"更强烈的一些词。）

他们决定修复时间问题，于是开始设计一个可以插入到从属处理器插槽中的小电路板。这样，整个电路板除了原来的微处理器以外，还在处理器与存储器之间引入了一个新的电路（参见图5-2）。这个电路板的设计和制作花费了很长时间，因为他们使用了手工布线原型设计，电路很复杂，而且连线上也有错误。最后，他们终于设法把这块电路板插到了处理器中并让它工作起来，以便观察几个月，看看问题是否得到解决。但问题依然存在。存储器仍然偶尔无法通过校验和检验。

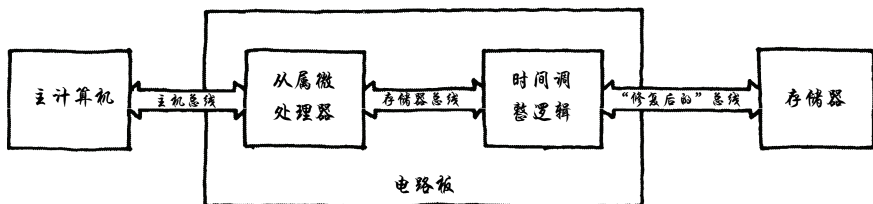


图5-2　初级工程师的解决方案

① 无粘接的接口，glueless interface，不局限于一种制式且不依赖于特定个体的接口。

我们的高级工程师很不赞成这种方法，因为没有人真正看到是哪里出错了。他坚持认为我们应该会看到进入存储器的数据会出错。他开始忙了起来，搬来了笨重的逻辑分析器，花了一番工夫把它接到系统中，并试着查看为什么数据会发生错误。实际上他并没有看到任何错误信号进入存储器，也很难查明数据是否正确，因为这些都是程序数据，它们看起来都是随机数据。于是，他写了一个规则的数据输入模式“00 55 AA FF”，用它来反复循环，作为输入数据。他希望看到类似于“00 54 AA FF”这样的错误，但实际上看到的却是“00 55 55 AA FF”。系统并不是写入了错误的数据，而是把正确的数据写了两次。

他回到主机总线这一端，用示波器测量了几个信号，发现在写入线路上有一些噪声。由于电路板上还有另外一个电路，因此它在数据写入过程中会产生短时脉冲波形干扰，这个干扰偶尔会很大，使得一个脉冲看起来就像是两个脉冲（参见图5-3）。

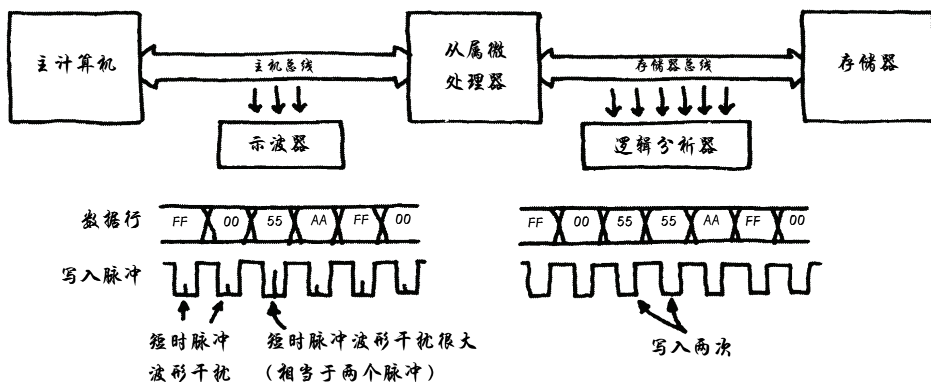


图5-3　高级工程师所看到的

在初级工程师所做的最初测试中，只是把数据写入寄存器，然后再读回来，这只使用了两个写入脉冲，写入两次，因此能够正确地读回数据。但是，当通过芯片下载数据时，每个写入操作都会把一个多余的字节发送到存储器中的下一个位置，使用两个写入脉冲意味着第二个写入脉冲将会把数据写到下一个位置，而

> 且它后面的所有写入脉冲都将依次顺延一个位置，从而造成“校验和”错误。我们由于找错了地方而浪费了几个月的时间，这完全是因为我们仅仅猜测失败的原因，而没有去观察它。■

亲眼看到底层的失败是非常重要的。如果你猜测失败是如何发生的，那常常会修复一些根本不是bug的问题。这样的修复不仅不会解决问题，而且还会浪费时间和金钱，甚至会破坏其他地方。请记住，不要这样做。

“不要想，而要看”，这是我最常跟工程师们说的一句话，比其他任何调试建议说得都要多。有时，当某位工程师提出了一个表面看来非常好的想法，但进一步研究发现它实际上根本算不上好想法的时候，我们就会开玩笑地说：“看，他是一位思想家。”所有工程师都是思想家。他们喜欢思考，这是一件有趣的事情，而且肯定胜过体力劳动，这也正是我们成为工程师的首要原因。虽然我们有各种各样好的设想，但发生问题的原因更加多样化，即使是最有想象力的工程师也无法想象出来。那么，为什么我们认为能够通过思考来找到问题呢？因为我们是工程师，因为想比看要简单得多。

观察是很难的。像上面例子中的那位高级工程师一样，必须把示波器和逻辑分析器接到电路中，这是十分困难的一步，特别是芯片很密集，无法只是简单地把探针夹到芯片上。你必须把分析器的线焊接到很小的引脚上并编写逻辑分析器程序，查明复杂的触发条件。（事实上，我曾认为把这个例子作为本章的开篇故事是不是太长了，但最终还是使用了它，因为它有助于说明问题。观察就算不是永远也是常常比你想象的要复杂得多。）在软件世界里，观察意味着设置断点、添加调试语句、监视程序值以及检查内存。在医学领域，需要测试血样和进行X线透视。它需要做大量的工作。

那种捷径（特别是以规则1作为借口）只不过是试图找出问题所在。

“噢，一定是rammafram，因为只是在我开启 frobnivator的时候它才出故障。”

“我运行了一个模拟测试，它工作良好，因此问题肯定不会出在那里。”

在内存问题中，“时间的设计很差。我们最好重新设计整个电路，这样问题就解决了。”

上面这些都是很容易（也十分常见）得出的结论（可能有关frobnivator的结论除外），看起来也提供了一种找到问题的简单方式，但实际上并非如此。

我曾经有位同事，他非常聪明，也对自己的逻辑思考和理解产品的能力非常自信。当他听说有bug时，总是会说："我敢打赌，它就是'这样这样'的一个问题。"我总是告诉他我跟他打赌。我们从来没有用钱来做赌注，这对我来说可真糟糕，因为几乎每次都是我赢。虽然他很聪明，也了解系统，但他并没有看到失败，因此没有足够的信息来查明原因。

当你的错误猜测一一被否定后，你精疲力尽，但你仍然必须找到bug。你需要做的工作量仍然跟先前一样多，唯一的不同就是你的时间变少了。这很糟糕，除非你认为"越早掉队，你就越有充裕的时间去追赶"。因此，为了帮助你在思考之前先进行观察，下面给出一些指导原则。

5.1 观察失败

如果想找到故障所在，必须真正看到发生故障的情况，这看似是显而易见的。事实上，如果没有看到失败，你甚至不会知道它已发生，不是吗？然而，这样说是不对的。当你发现bug时，你看到的其实是失败的结果。比方说，我打开了开关，灯没有亮。但实际的问题出在哪里呢？是开关坏掉了致使电流无法通过，还是由于灯丝坏了而使电流无法通过？（或者仅仅是由于我按错了开关？）你必须仔细观察，找到足够多的问题细节，才能调试它。在从属处理器的例子中，那些初级工程师虽然发现了保持时间很短，但也并没有据此去观察被错误写入的存储器。但即使他们观察了，也不会发现错误，因为错误并不是发生在那里。

如果你不能留意实际情况发生的全过程，那么你极有可能曲解很多问题。你猜测某个地方出了问题，于是修复它，但实际上错误发生在另一个地方。由于你没有看到一个字节发生了改变，导致用错误的参数调用了一个子例程，或者一个队列溢出，而你却去修复了一个完全没有发生错误的地方。这样，你不仅没有修复问题，而且还可能改变了时序，因此把问题

隐藏起来了，这会使你误认为已修复问题。更糟的是，你可能会破坏其他地方。即使在最好的情况下，这也会导致时间和经济上的损失，就像一个（打不好高尔夫球的）人去买一套新的球杆，而不是请职业高尔夫球手帮他分析击球的姿势。新的球杆不会帮助他纠正总是把球打偏的问题，而高尔夫球的课程很便宜。他仍会打出很多次双柏忌[1]，最后不得不放弃尝试，去参加高尔夫球课程。



案例故事　我的老板曾经帮过他的邻居一个忙。这位邻居是卖水泵的，他觉得欠了老板一个人情，就许诺说："如果你什么时候需要一个新水泵，一定来找我，我将为你安装一个最好的水泵。"有一天，我的老板出差了，他的妻子听到一阵发电机的声音，大概持续了十几秒钟，然后就停止了。这种声音只是偶尔响起，每隔几个小时就出现一次。由于丈夫不在家，她就请邻居来看看是怎么回事。"是水井泵出问题了！"邻居回答说，并答应第二天解决问题。第二天，他的施工队取出了旧的水泵，又安装了新的，然后离开了，他完成了出色的工作，也还了一个人情。但是拆换水泵把井底的沉淀物都搅起来了，因此他们接下来的几天就要忍受混浊的井水和散发出的氯气了。而且，发电机的声音还是时不时地响起。

那天晚上，我的老板给她的妻子打了电话，知道了这次变故。"是什么使你认为问题出在水泵上呢？是水压变低了吗？""不是。""地下室的地面上有积水吗？""没有。""有没有人站在水泵旁边听到它发出噪声？""没有。"

最后大家才发现事实真相，原来，在我的老板离开之前，他用车库里的电动空压机给车胎充了气。当他离开时，没有把空压机关上，当空气从软管中漏出时，电动机时不时就会启动，把损失的气压补回来。那台旧水泵并没坏，虽然更换它并没有花老板的钱，但他接下来就得处理沉淀物和氯气的问题。我猜想他的妻子和那位邻居接下来也要应付老板对他们的调试技术的评论了。■

① 双柏忌，double-bogey，总杆数高于标准杆数两杆称为双柏忌。

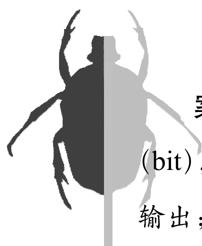


案例故事 我的一位同事告诉我，他们公司有一台服务器每天夜里几乎在同一时间发生崩溃并重启。他们记录了重启的日志，但从所有记录上都看不出问题的原因。他们试着监控自动运行的进程，因为他们觉得既然服务器几乎总是在同一时间发生失败，问题也一定是由某种自动运行的东西导致的。但经过了几个星期的监控，没有发现任何看起来有关联的问题。于是，我的朋友决定晚上留在办公室，看看机器到底发生了什么问题。11点刚过，机器电源突然断了。他回头看了看，发现大楼管理员刚刚把服务器的电源从插座上拔出来，他是想“借用”插座使用他的吸尘器。大楼管理员认为这样做没有问题，因为几星期以来他每天都这么做。其实问题是很显然的，只要有人实际看到了问题的发生，就会发现它。■

一定要亲眼看到实际错误是如何发生的。观察往往比猜测能够更快地找到问题。因为猜测虽然看起来是捷径，但这条捷径并不会带你找到问题的根源。

5.2 查看细节

上面所讲的例子是一种极端情况——在观察上所花的工作量小到极点（尽管如此，故事中的主人公却没有在一开始的时候就去查看问题）。更典型的情况是，每次为了发现故障而观察系统，都会了解更多与失败有关的信息。这将帮助你确定应该进一步观察哪些地方以获取更多细节。最后，你会得到足够多的细节，这时才可以根据这些细节来查看设计并找到问题的原因。



案例故事 我们正在使用一个视频压缩软件，它可以把视频压缩成很小的位（bit），然后从一个地方传输到另一个地方。如果工作正常，可以得到很好的视频输出；如果有错误，画面就会有花屏而且看起来很奇怪。我们的视频质量比预计

的要差得多，花屏现象较为严重，因此我们推断一定是某个地方出了问题。

现在，视频压缩使用很多技术来节省位。它不是发送每一帧（每秒钟有30帧）中的所有像素（点），而是消除冗余的信息。例如，如果背景没有改变，它会发送一个“无改变”位，这样接收端只需重新显示前一帧的相同背景即可。视频压缩有数十种不同且互相关联的压缩机制，它们都会影响视频的质量，而且很难分析和纠正。如果对于错误没有很好的认识，那么只能靠猜测和编码，这样要想提高视频质量可能要花费数月时间。

有一种称为“运动估计”（motion estimation）的压缩技术，它搜索画面的某个部分（例如我的手）在下一帧中是否移动到新的位置（例如当我挥手的时候）。（视频压缩人员为了查看压缩效果，经常会做挥手的动作。）如果发现移动，它可以用很少的位把新的画面表示出来，基本上只需表达这样的信息：“画面的这个部分与上一幅画面相同，只是把X像素移开，并把Y像素向上移动。”（参见图5-4）。至于这个部分是什么样子的，并不需要描述，因为在背景中，接收端已经从上一幅画面知道了。如果它无法发现移动，那么就必须完全重新描述手的样子，这需要很多字节，而且生成的画面质量也较差。

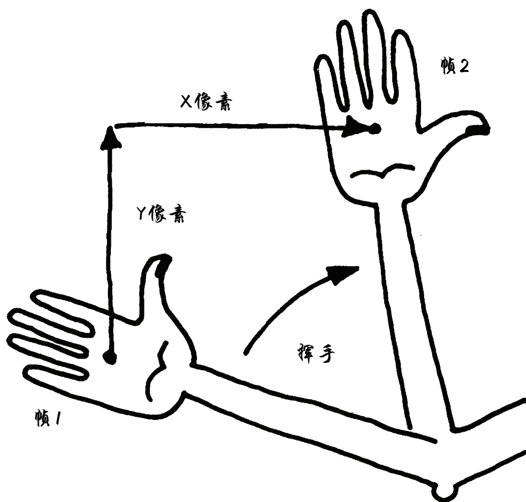


图5-4　运动估计

在我们的例子中，移动的对象看起来是最差的部分，因此我们决定深入研究一下运动估计。但我们并没有试图去优化代码，甚至没有去分析它，而只是想要看看系统是否能够发现移动的对象。我们使用了一个分析软件，它可以在输出屏幕上把检测到的运动显示为小的方块，我们用颜色来表示方向，用亮度来表示移动的速度。现在，当我的手向下移动时，我看到屏幕上显示出橙色的小方块，当我加快移动速度时，方块的亮度变大。当我向上移动时，我看到了紫色。当我向左或向右摆手时，我看到绿色和蓝色的小方块，但数量很少。

我们奇怪为什么系统没有检测到左右的移动，于是把详细的运动计算快照输出到一个单独的调试监视器上，包括在所有搜索位置上的画面匹配情况。我们很惊讶地发现软件只搜索很少一部分水平位置，而跳过了其他部分。匹配算法并没有问题，没有匹配的原因在于搜索算法没有查找所有可能的位置。我们修复了这个简单的bug，之后检测到大量的水平动作，画面质量也有了很大的提高。■

在停下来思考问题之前，对细节的观察应该到什么程度才合适呢？简单的答案是：“一直观察，直到把问题的原因锁定在几种可能性之内。”在前面的例子中，一旦发现搜索的范围不完整，我们就查看搜索代码并很快找到了bug。我们看到的问题是搜索算法没有涵盖所有位置。那么，我们是否应该查所有代码来查找错误呢？不，搜索是由一个很小、很简单的软件例程控制的。那么我们是否应该在看到水平移动没有被检测到的时候就去查看代码呢？当然不能——搜索失败有很多种原因，甚至需要查看代码的每个地方。这会浪费大量时间来检查代码，以确定画面是否匹配，这样我们就不会把重点放在研究搜索位置上。因此我们没有在那里停下来，而是继续深入研究足够多的细节，确定原因是搜索问题，而不是匹配问题。

在水井泵的例子中，他们甚至没有到地下室去听听声音就开始解决问题了。如果他们先到地下室去查一查，就会听到声音是从车库中的空压机传来的，而不是来自水井泵。他们只是凭猜测去行动，但有很多机器都会发出那种声音。

经验可以起到帮助作用，就像理解系统会起到帮助作用一样。当你作出了错误的假设并沿着它追查问题时，经验会告诉你在特定情况下追查到什么程度就应该停止了。经验会告诉你什么时候问题的原因已经被锁定到一个很小的范围内。你会知道怎样做一个好的调试人员。评判标准不是多快地提出一个猜测，也不是猜测得有多好，而是尽可能少地按错误的猜测行动。

5.3　问题忽隐忽现

在调试间歇性bug时，观察底层的失败细节有另外一个好处，这在前面已经讲过，这里再重申一下。看到底层的失败细节后，当你认为已修复bug时，很容易证明确实已修复。你不必依靠统计数据，就可以看到错误不再发生。当前面例子中那位高级工程师修复了从属处理器上的噪声问题时，他可以看到写入脉冲中的短时脉冲波形干扰消失了。

5.4　对系统进行插装

既然你已经决定观察系统，那么就应该采取一些观察措施。你需要把工具植入到系统中，或连接到系统上。最好的做法是植入到系统中，在设计期间就植入一些能够帮助你观察内部行为的工具。既然bug就是在这时植入的，那你当然应该同时可以植入调试工具。但是在设计的时候，你无法预料到调试时需要看到的每件事，所以会漏掉一些事情。这就要求在调试时构建特殊版本的系统，以便把工具插装到系统中，或是添加外部的插装工具。

5.4.1　设计插装工具

在电子硬件领域，这意味着设置测试点。添加一个测试连接点，以便于观察总线和重要的信号。最近，由于人们使用了可编程的门阵列（gate array）和专用的集成电路，问题往往隐藏在逻辑块的内部，我们无法把外部工具插装到这些逻辑块中，因此从芯片输出的信号越多，就越容易找到问题。把所有寄存器都设计成可读、可写的。添加LED和状态显示器，它们可以帮助你研究电路的内部。你是否注意到在一些个人电脑上运行系统状态软件时，可以

告诉你主处理器的温度？这是因为设计者植入了温度传感器，因为处理器一般是封闭在机箱内部的，所以这是唯一能够告诉你处理器是否过热的方法。

在软件领域，最初级的内置插装策略通常是以调试模式编译，这样就可以通过源代码调试器来观察程序的运行。遗憾的是，当程序正式上市时，就必须以发布模式来编译了，这样就无法再用源代码调试器来调试产品代码了。因此，你必须采取第二个选项（并非辞去工作而去当一位摇滚歌星），就是在性能监视器中输入各种有意义的变量，以便在运行时观察它们。在任何情况下，都应该开启一个调试窗口，并且让代码输出状态消息。当然，这个窗口应该能够把消息保存到调试日志文件中。

收集的状态消息越多，就越有利于调试，但应该有某种方式来控制选中消息或消息类型的开启和关闭，以便为了调试特定问题而重点查看所需的信息。此外，把消息输出到调试窗口通常会使系统发生一些改变，从而对bug造成影响。（一种常见的情况是启用调试器会极大减慢系统的速度，以至于bug不再出现了——这就是它们被称为debugger的原因。参见5.5节。）如果把过多的消息发送到调试窗口，可能也会极大地影响系统处理器的速度，当每次鼠标点击都要花费35秒时，你会感觉无法忍受。

状态消息的开启和关闭有3种不同级别选择：编译时、启动时和运行时。在编译时开启状态消息可以节省编码工作，但一旦产品发布之后，就无法再调试了。在启动时开启状态消息很容易实现，但一旦系统开始运行之后，也无法再调试。在运行时开启状态消息会增加编码的难度，但它是最灵活的选项，因为可以在任何时候进行调试。如果在启动时或运行时开启状态消息，甚至可以告诉客户如何开启状态消息，并进行远程调试。（这自然要求我们确保调试语句拼写正确，没有淫秽内容，没有政治立场不正确或反动的内容。）

状态消息的格式对后续的分析工作将产生很大影响。把消息分成各个字段，这样特定的信息总是出现在特定的栏中。用一栏来记录系统的时间戳，它应该精确到足以调试时序问题。还有很多标准的栏可供选择，包括消息是由哪个模块或源文件输出的；消息类型的通用代码，如“info（信息）”、“error（错误）”或“really nasty error（严重错误）”；输出消息最初是由哪位工程师写的（为了跟踪谁做了什么工作，以及他为什么要输出这些消息）；运行时数据，

例如命令、状态码和预计值与实际值的比较，这些能够为你提供后面的调试工作所需的详细信息。最后，采用一致的格式和关键词也有助于在后续的调试工作中过滤调试日志，从而帮助你专心查看真正需要的数据。

在嵌入式系统中（计算机没有显示器、键盘或鼠标），软件插装需要添加某种输出显示：一个串行端口或一块液晶显示板。大多数DSP都有开发端口，可以从一台单独的PC机来监视操作系统的实时运行。如果嵌入式处理器是植入计算机中的，则可以使用主计算机处理器的显示器，并在嵌入式处理器和主计算机之间添加通信机制，例如共享的内存位置或消息寄存器。为了在没有实时运行操作系统的情况下查看代码时序问题，可以添加一些硬件信号，并且在进入例程和退出例程时可以开启或关闭这些信号，这样就可以用硬件示波器来观察这些信号了。植入电路中的模拟器为我们提供了一种跟踪代码行为的方式（代码并不知道你正在跟踪它的愚蠢或荒唐的行为）。

在使用嵌入到电路中的模拟器的时候要十分小心，虽然它们是很好的软件调试工具，但它们与插入的硬件处理器完全不同，这一点是众所周知的。它们不仅存在时序和内存映射上的差别，而且有时整个功能都会丧失。因此，不能用模拟器来验证嵌入式处理器电路的硬件设计。然而，一旦你设计了可以解决模拟器误差的电路，那么它是插装到嵌入式电路软件中的最佳工具。

最基本的原则是从设计一开始就考虑调试的问题。一定要把插装作为产品需求的一部分，并且把插入工具的接入方式写到每个功能规格和API定义中。标准的实用工具集中必须包括调试监视器和分析过滤器。这些做法会为你带来额外的好处，它们不但使得最后的调试过程变得更简单，而且当你思考哪里需要做插装时，这还有助于更好地设计系统并从一开始就避免某些bug。

5.4.2　过后构建插装

无论在设计时考虑得多么周到，当开始调试时，都必须面对一些无法预料的情况。不必担心，你只需在必要的时候对系统进行插装即可。但有一些注意事项。

在对系统进行插装的时候，一定要确保起始的设计环境与发现bug时的环境相同（不要模拟失败），然后再增加你所需的插装工具。这意味着使用相同的软件和硬件环境。植入插装工具后，要使失败再次发生，以便证实环境确实相同，而且插装工具没有对问题造成影响。（参见5.5节。）最后，当找到问题后，解决问题并清除所有插装，以便不影响最终产品。（当然，应该保存一个副本，以备将来需要——把有错误的代码注释掉或加上“#ifdef”标记，而不是单单删除它。）

临时插装的好处在于它能让你看到错误是如何发生的。



案例故事　我有一个可编程的门阵列，它的行为一度很反常，于是我对它进行了重新编译，然后使用几个空闲出来的外部针作为示波器的探针。每次想要观察一个新的信号时，都需要重新编译这个阵列，但我确实看到了我需要观察的东西，并解决了问题。■

程序的原始数据形式往往不便于分析。这正是插装工具的用武之地，它可以对数据加以整理，使得所需的细节变得更明显。



案例故事　我们有一个通信系统，它的问题是在经过几个缓冲阶段后，会破坏数据。我们并不知道数据是被改写了还是删除了，因此添加了一些调试语句，把内存缓冲器的指针值输出来。指针是很大的十六进制数，而我们怀疑出问题的地方正是它们之间的空间的大小，因此我们加入了一种计算，用于确定指针之间的差别（包括缓冲结束的绕回编址），并把它输出来。这样，我们很容易就看到读指针有时会突然提前一小段时间执行读操作，而此时它要读取的数据尚未被写入。我们只分析了一小部分对指针进行操作的代码就发现了bug。■

那么，在调试时应该查找一些什么信息呢？你所选择的那部分内容应该能够证实你的判断，或者显示出你未意料到的行为（正是这些行为导致了bug）。在下一章中，我们将给出一些更详细的搜索技巧，但现在，关键是获取有关的细节。观察变量、指针、缓冲层次、内存分配、事件时序关系、信号标记和错误标记。查看函数调用和退出，以及它们的参数和返回值。查看命令、数据、窗口消息和网络数据包。获取详细信息。在运动估计的故事中，我们通过输出搜索位置和匹配率的信息，发现了搜索算法的错误。

5.4.3　不要害怕深入研究

我以前曾经看到过一些对软件成品（即已发布的软件）进行调试的建议："由于你无法修改软件，而且它没有源代码调试功能，因此你应该使用现有的API，依次测试各个模块，以便隔离有问题的模块。"我并不喜欢这条建议。它违背了"不要模拟失败"这条规则，预先假定了你可以方便地用API来测试各个模块；而且，即使你顺利地隔离出有问题的模块，也没有办法进一步查看这些模块，因此你只能想，而无法去看。

如果代码中有bug，为了修复它，你需要重新构建软件。首先，你会为了发现bug而重新构建软件。我们可以构建一个调试版本，以便能够查看源代码。添加新的调试语句来查看真正需要查看的参数。"不要想，而要看"，然后，在修复bug后，用"#ifdef"标记所有调试语句并重新交付产品代码。

前面曾提到过让工程师带着插装工具，乘坐出租车前往客户现场。由于客户现场通常只有软件成品，而且工程师很难在客户那里构建软件，因此他们经常需要使用附加组件或已经植入系统中的插装工具。但当这些工具不足以找到bug时，就需要在你自己的实验室中建立一个模拟系统，以便用更多工具对软件进行测试。当然，你必须保证系统在实验室中发生失败，这样才能确定你的模拟是正确的。如果系统在实验室中没有发生失败，就需要在现场为软件添加插装工具；派工程师带笔记本电脑和调制解调器前往阿鲁巴岛，然后在实验室里构建他们需要的东西，再用电子邮件发送给他们。

5.4.4 添加外部插装

如果你不想或无法植入内部插装工具，那么至少应该添加外部插装工具。当调试硬件时，可以使用量表、示波器、逻辑分析器、光谱分析仪、热电偶或其他用于观察硬件的设备。如果要调试PC机内部的问题，就必须在主板上连接各种测量仪器。此外，所有设备必须具有足够快的速度和精确度，以便能够测量到错误。低频示波器无法找到高频问题，数字逻辑分析器无法发现噪声和短时脉冲波形干扰。用手指就可以知道芯片是不是热得都摸不得（不要愚蠢地这样做），但这并不会告诉你芯片是否是由于过热而导致运行错误。

当调试软件时，如果你无法使用调试器来查看内部代码，那么有时可以接入一个用来调试总线的分析器，当机器执行指令时，它可以对这些指令进行反汇编。由于你只能用汇编语言来进行调试了，因此这是最后的办法。除非你是像“航海铁人”[①]那样的顽固分子，认为汇编语言早已过时。（警告：你知道ASCII码，可以做十六进制数的加减法，你的确关心进位的状态。）



案例故事 我们过去曾经使用一台VCR作为一个外部插装工具来调试录像显示问题。问题看上去似乎是系统没有按照正确的顺序来显示各个帧，我们把录像录了下来，然后使用VCR的一帧一帧播放功能来播放录像，结果发现系统实际上把相同的帧显示了两次，而电视的隔行扫描使这看起来就像是录像被倒放了。■

5.4.5 日常生活中的插装

在医学领域，我们用体温计来测量体温，用X光透视来诊断癌症。心电图仪（用于测量心脏内部的电信号）有一个探针，它看上去和逻辑分析器的探针一样，而且它们可能使用了同样的塑料壳。我们必须使用这些外部仪器，因为已经没有机会在设计阶段往人体内部植入

① 航海铁人，wooden ships and iron men，一款海战游戏的名字。

内置插装工具了。但是现代医学正在发现一些已经内置好的插装。例如，遗传疾病的标志性基因或预示着前列腺癌的化学物质的出现。（在这种情况下，使用内置的插装工具比用数字探针更有效。）

当水管工人在锅炉中安装水温计或在水箱中安装水压计时，就相当于在系统中进行插装。（在水井泵的故事中，如果他们检查一下水井的压力计，就会看到水泵没有问题。空压机也有一个压力计，它的压力会缓慢地下降，而当开动时，压力会迅速升高。）

为了查找房屋漏空气的地方，我们可以拿着一个丝带靠近窗户和电源插座，检查有没有气流；如果你能负担起的话，可以使用红外线传感器来查找温度较低的位置。当自行车轮胎漏气时，我们可以把肥皂水涂抹到车胎上，并查看气泡。（这是假设在车胎上找不到内置插装——明显的钉子——的情况下所采用的办法。）我们可以用肥皂泡来检查后院烤肉架的燃气罐是否漏气，而不想使用内置的插装来检验（剧烈的爆炸），因为那些供烧烤的原料本身就有很大的味道（抱歉）。天然气中加入了臭鸡蛋气味，目的就是当泄漏时能够被发现。为了找到古代硬币和发卡，用金属探测仪来搜索海滩显然比随意地挖掘要好得多。

> “身在其中，方知其味。”
>
> ——西西里岛谚语

5.5　海森堡测不准原理

海森堡是量子物理学的开拓者之一。他致力于研究质量和体积极小的原子内的粒子，他发现你要么测量一个粒子的位置，要么测量它向哪个位置运动，但这二者当中有一个测量得越精确，另一个就越测不准。无法得到准确测量的原因是探针成为了系统的一部分。换言之，测试工具影响了被测系统。

前面已经讲过了调试器对时序的影响。任何插装都可能对系统造成影响，只是程度不同而已。示波器的探针增加了电路的电容。软件的调试版本影响软件的运行时间和代码的规模。在PCI总线上增加扩展卡改变了总线的时序。甚至打开机箱盖子也会改变内部零件的温度。

我可以非常肯定地说，这是不可避免的。你必须记住这一点，这样就不会感到意外。此外，一些插装方法的干扰性要小一些。用声纳来探明矿藏比在产煤国家挖地三尺找矿更有利于保护环境。X光透视或CAT扫描比探查性手术的破坏性更小，但结果可能不够精确。

正如第4章中所提到的，即使微小的改变也可能对系统造成足够大的影响，导致bug被完全隐藏起来。插装就是这些改变之一，因此在为有故障的系统添加插装工具之后，要使系统再次失败，以证明你没有为海森堡问题所困。

5.6 猜测只是为了确定搜索的重点目标

“不要想，而要看”并不意味着不能做任何猜想。事实上猜测是好事，特别是当你理解了系统之后。你的猜测可能很接近事实，但猜测只是为了确定搜索的重点。在尝试修复问题之前，仍需要再次看到失败，以便确认你的猜测是正确的。在视频压缩和运动估计的故事中，我们猜测运动估计可能出了问题，因此开始进行运动检测。当我们左右挥手而看到屏幕上只出现很少的蓝色和绿色的小方块时，我们可以确认猜测是正确的。然后，我们猜测匹配逻辑没有起作用，并查看了计算；当我们看到搜索算法有错误时，这个猜测就被否定了。

因此，不要过分相信你的猜测，它们往往偏离了方向，并且把你引入歧途。如果事实表明，经过仔细的插装仍然无法确定你的猜测是否正确，那么就到了退回并再次猜测的时候了（要么在占卜板上重新再占一卦，要么把bug原因依次写在飞镖盘的各个格子里，然后投掷一次飞镖看看命中了哪个原因。你可以选择你自己的方法。我建议你采用规则4）。在从属处理器的例子中，高级工程师首先查找了被写入存储器的错误数据，但他并没有找到。然后他设置了一组重复的数据，并把注意力调整为搜索微处理器的主机端，在这里他发现了被写入两次的脉冲。

有一个例外：之所以会按照某个特定思路进行猜测，那是因为某些问题比其他问题更容易出现，或者比其他问题更易于修复，因此首先检查这些问题。实际上，当你猜测是某个易发生且易修复的问题时，只有这时，你才应该不用真正看到失败的细节而直接尝试修复它。在前面的例子中，我打开了开关，而灯没有亮，我推测可能是开关坏了，也可能是灯泡坏了。但灯泡

坏的可能性更大，而这很容易修理，只需换个灯泡即可，如果新的灯泡亮了，问题也就解决了。但我敢打赌即使在换完灯泡后，你仍然会摇动旧灯泡，只是想证实里面有一根灯丝断掉了。



案例故事　我的一位朋友有一台安装在水箱中的电热水器。它有一个内部电加热器，有一天水箱突然不出热水了。他拨通了服务热线，被告知可能是内部保险丝坏了。于是他出去买了一个保险丝（一个看起来怪模怪样的6英寸的金属线），在狭窄的空间里，费了好大力气才把旧的保险丝换下来。但是仍然没有热水。最后查明问题是一个电路断路器跳闸了，这个问题只需几秒钟就可以修复——打开断路器盒盖，找到跳闸的断路器。我的朋友不仅猜错了解决办法，而且他的猜测也是较难实现的一个。他从未告诉我买保险丝花了多少钱。■

5.7　小结

不要想，而要看

凭空想象，问题可能有几千条原因。而实际的原因只有去看了才能发现。

- **观察失败**。高级工程师看到了真实的问题，并且能够找到原因。而初级工程师们认为他们知道错误发生在哪里，结果他们修复的地方根本没有出错。
- **查看细节**。听到水泵似乎发出声音时不要停下来。到地下室查明是哪个水泵。
- **植入插装工具**。使用源代码调试器、调试日志、状态消息、信号灯和臭鸡蛋的气味。
- **添加外部插装工具**。使用分析器、示波器、量表、金属检测仪、心电图仪和肥皂泡。
- **不要害怕深入研究**。虽然它是软件成品，但它出问题了，你必须打开并修复它。
- **注意海森堡效应**。不要让仪器影响了系统。
- **猜测只是为了确定搜索的重点**。大胆地猜测内存时序发生了错误，但在修复之前应该先查看它。

第 6 章

分 而 治 之

“当你排除了所有的不可能，不管留下了什么，也不管看起来多么不可思议，那必定都是事实”。

——福尔摩斯，《四签名》



案例故事 这个故事讲的是一家滑雪场的旅店预订系统，当时这是一个最先进的系统，它使用Macintosh机器作为前台，并通过网线连接到后端的LISP数据库服务器。（说到LISP，我们就不能不想到它的发展历程。你知道，它是第一个应用在人工智能上的语言，经历了1985年的“未来之风”，又在1989年与我们挥手说再见，但至少Macintosh曾经掌控了个人计算机的世界。我这么说并没有挖苦之意。）

滑雪场员工抱怨说，当他们从数据库机器上读取数据时，Macintosh终端机的速度变得越来越慢。其中有一台终端特别慢（而且总是如此），有时无法完成数据库查询，并显示一条错误消息。有一位技术人员被派去解决这个问题，在午夜的时候（旅馆在夜间也照常运营），他对问题进行了研究。他了解了这个系统：运行数据库的机器和终端机通过串行线进行通信，他检查了传输出错时的信息。如果出现一个错误，计算机将重试，直到获得正确数据，或者在长时间无法获取正确数据时报告一条错误。这位技术人员猜想系统的通信发生错误（他的猜测是正确的），因此导致系统需要长时间地重试数据的传输。他通过查看通信软件中的调试信息确认了这一点，这些信息显示出在双向传输时都有错误并重试过。由于系统

一直在正常运行，因此软件显然没有发生改变，所以他猜测问题出在硬件上。

数据库机器中有一块特殊的电路板，它通过一条扁平带状电缆引出了8条通信电路。电缆进入安装在墙上的一个“转接盒（breakout box）”，转接盒前端有8个串行线接头。转接盒的作用是为接头提供空间，因为数据库机箱后面没有足够的地方安装这8个接头。从转接盒引出的8条串行电缆直接连到Macintosh终端上（参见图6-1）。

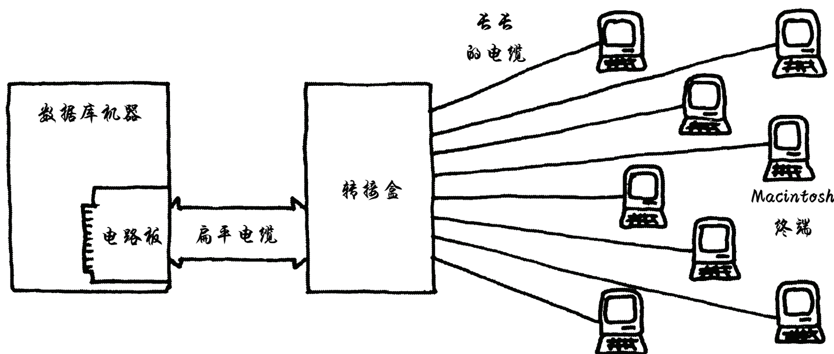


图6-1　旅店预订系统

技术人员并不知道问题是出在数据库计算机的电路板上，还是出在终端连线上，但由于所有终端运行都不正常，所以他认为问题不在终端。他用示波器观察电路板与线的连接处，也就是扁平电缆连接计算机的地方（终端与数据库机器之间不断有“Are you there?”的消息，因此他查看了进入和发出的信号）。他发现发送给终端的信号很强，很好，但进入的信号却很弱，很差。

这说明复杂且昂贵的电路板并没有出错，他松了一口气。但问题很可能出在一向可靠的电缆上，这令他很诧异，由于连接终端线路的“中转站”（即转接盒）比较便于观察，于是他在这里查看了8条串行线在接入转接盒处的信号。这次的结果正好相反，来自终端的信号很好，而转接盒发出的信号很差。他从调试信息知道两个方向的传输都有问题。现在，他发现问题就发生在他正在观察的两个硬件点之间。于是他选择了一个折中点，查看了扁平电缆与转接盒的连接处，结果再次相反：发出的信号很好，而进入的信号很差。问题似乎是出在转接盒上。为了证明这个猜测，

他测量了转接盒从串行电缆接头到扁平电缆接头的电阻（也就是他测试的最后两个点之间的电阻），发现阻值很高，而这是不合理的。于是他打开了转接盒。

转接盒中除了被焊接到电路板上的接头以外，什么也没有。他在线路的不同点之间测量了电阻，发现串行线接头的引脚与电路板的焊点处阻值很高。他仔细查看了焊接处，看上去每个引脚周围都有一些像头发丝一样细的裂缝。他用电烙铁把每个引脚又重新熔化并焊好。之后阻值也恢复了正常。处理完所有的引脚后，他插上了线，并确定终端都能工作了。然后他拔下了所有插头，把盒子重新盖上，又把所有插头都插了回去，并再次确认终端都能工作。这样做是符合Goldberg所提出的“墨菲定理的推论”的，这个推论讲的是这样一个定律：重新装配是绝对必要的，如果不这样做的话，当你再次测试的时候有可能证明问题并没有修复，这样你就需要重新拆开所有东西，导致重新装配的工作量更多。

所有终端都能正常工作，除了那台特别慢的终端以外，它的速度仍然很慢。

他重新泡了一杯咖啡，再次查看最慢的那台终端的新的调试记录。这些记录显示出发送到终端的数据发生错误，而由终端发来的数据则没有错误。他打开了串行电缆与转接盒的接头，并再次观察信号。流出的信号看起来良好。他向“下游”移动，打开了连接终端的接头，查看进入终端的信号，意外地发现有几条线竟然没有接上。

他查看了线路图，图中显示电缆共有6条线，其中有两条线用于输入信号，两条线用于输出信号，还有两条线是空闲的。（由于6芯电缆是最便宜也最容易买到的，因此即使浪费两根线也要比买4芯电缆划算。）连接电缆的那个人可能没有查看线路图（或许接线的时候灯光昏暗，或许他是色盲症患者），他在一端没有接紫色的线，而是接了蓝色的线，但在另一端却正确地接了紫色的线。蓝色的线和紫色的线虽然没有接到一起，但它们在同一条电缆中行进了数百尺，它们之间产生了足够的信号耦合，使得终端可以工作，虽然有时效果极差。当这位技术人员把两端都接上紫线时，终端完全正常工作了。■

在这个故事中，我们能够发现很多调试规则。技术人员“理解了系统”，并在此基础上集中检查硬件。然而，他没有简单地替换所有硬件，因为他“没有想，而去看”。（当然，任何人都无法猜出终端速度慢的原因是线没有接对。但反过来，任何人都会知道这样的接线错误会导致终端无法工作。）他使用了内置插装来检测通信错误以及错误的方向。他使用系统之间的常规通信来“制造失败”，因此能够用示波器看到问题。他通过测量焊点的电阻发现了实际的问题，并且在重新焊接后测量了电阻，从而确认问题已修复。而且，当接线看起来有问题时，他又查看了接线。

但这位调试人员在这里所演示的最好规则却是“分而治之”。他通过反复地把问题分成好的一半和坏的一半，来缩小搜索范围，然后进一步研究有问题的那一半。

首先，他把系统分为软件和硬件两部分，由于他知道问题随时间变得越来越严重，因此他猜测问题出现硬件上。然后，他查看了转接盒的硬件信号，并观察到信号非常弱，这确认了他的猜想——硬件有问题。然后，他继续观察信号，每次都看到有一端的信号很差，于是他沿着错误的那个方向进行搜索。当他查出信号较差的线路时（现在是在计算机这一端），接下来他又沿着这个线路回到上一个测试点。甚至当他打开转接盒时，仍然用量表准确地定位哪里的连接阻值高。（参见图6-2）。

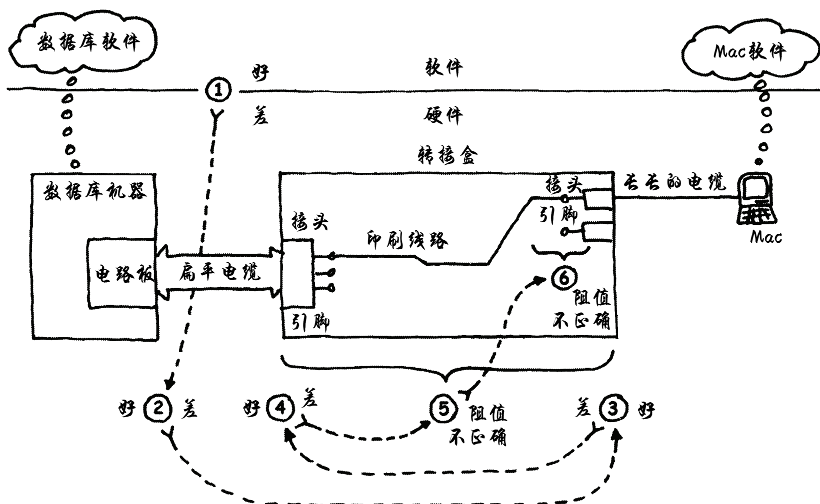


图6-2　查找问题的步骤

6.1 缩小搜索范围

你可能已经注意到，在查找问题时，“分而治之”实际上是第一条需要使用的原则。事实上，在查找问题时它也是唯一需要应用的规则。所有其他规则都只是帮助你遵循这条规则。分而治之是调试的核心，很多人都知道它，但很多人都没有遵守它，这也正是我写本章的原因。

缩小搜索范围，向目标追踪，找到目标范围。任何有效的目标搜索都会使用一种共同的技术，那就是“逐次逼近”（successive approximation）。我们希望在某个可能范围内找到问题，因此从范围的一端开始，先搜索前一半，看看是否有错误。如果有错误，则把搜索范围定为前四分之一，然后再次尝试。如果没有错误，则把搜索范围定为后四分之三，然后再次尝试。每次搜索都会查明目标的方向，每次搜索都会缩小一半的范围。在几次搜索之后，你就会找到目标。

让你的朋友从1～100中选一个数字，由你来猜，每次你的朋友告诉你是猜高了，还是猜低了，只需7次你就会猜到答案。如果这个数是42，那么你猜测的顺序可能是50、25、38、44、41、43，最后得到42。如果bug的位置可能有100个地方，那么你希望只用7次就找到它，而不是100次或50次，甚至也不是20次。软件搜索算法利用这种方法来扫描大的数据库，而无需永久地等待下去。在硬件中，我们使用高速的模拟-数字转换器来测试输出值，以便排查输入电压，在这个过程中，我们从高阶位开始，向低阶位进行排查（每次排查都会把前一次排查的路径缩短一半）。我猜想过去船上的炮手就是利用这种逐次逼近的方法以最快速度来瞄准目标的。我们在解决问题的时候，都希望能够快速完成任务。



案例故事 我曾经在电话公司工作过几个夏天，我工作的地方叫做“配线室”。来自各个房屋的所有电缆都从这里通向大楼，电缆连接到“立柱”上，立柱由大约100条线组成，每条线上又有数百个接头。立柱后面是“横架”，架子的前面一侧排列着接头，它们连接到中央机房的电话交换设备上。要想把一所房屋连接到

一台交换机上，先从立柱电缆接头接出一对儿线，然后把它引到正确的横架上，再沿着横架接到正确的交换机接口上。任何电缆可以与任何交换机相连。这是一种老式的技术，但是很有效。

当然，电话公司需要跟踪哪所房屋连到哪台交换机上，偶尔线会被接错地方（有时只是因为雇不到好的暑期帮工）。现在，假设你从立柱上拿下一根线，想要弄清楚它连接到横架的什么地方。

第一步是观察当线到达横架时走的是哪条路线。然后把线的一端交给助手，你绕到横架这一侧，沿着线的方向走到横架的中间，然后把手臂伸到沿着横架布设的线捆中（大概有数千条），同时你的助手拉动线的一端，造成线的运动，你则在这边感觉有哪根线动了。通过这种方法，很容易找到被拉动了的那根线；如果无法找到，则说明线的运动传递不到这么远的距离，于是你再向原路返回一半。一旦你感觉到被拉动的线，就抓住它，这时你变成“yanker”（拉动者），而你的助手则变成了“feeler”（感觉者），然后再次移动一半的距离。这样一段一段地重复这个过程，你们两个人就能够找到这根线最终被接到了横架的什么地方。这是个标准的过程，只是我并没有使用“yanker”和“feeler”作为正式的术语，如果我这样使用的话，肯定会招致大家的一致不满[①]。■

逐次逼近依赖于两个重要的细节：你必须知道搜索范围；当你查看一个位置时，必须知道问题在这个位置的哪一侧。如果你猜1与100之间的一个数字，而你的朋友选择了135，或者不告诉你猜测的数字是高还是低，或者说谎，每次都改变答案，那么你就不会猜中。（你应该换个朋友来做这个游戏。）

6.1.1 确定范围

如果你把整个系统作为搜索的范围，那么范围的确定就很容易。这可能比你实际需要的

① yanker，在这里是指拉动电缆的那位助手，yanker这个词最初曾被用来形容英俊的美国青年，但后来有人认为它带有贬义，类似于“美国佬”，所以作者说没有用它来作为正式的术语。

范围大得多，但每次猜测都能够缩小一半，因此从这个范围开始搜索还不算太坏。我们的技术人员从整个系统开始，并通过第一次猜测把范围确定为硬件，而把软件排除在考虑范围之外。如果他猜测电路板出了问题，并且把范围定在这里，那么他的搜索范围最后就被限定到扁平电缆上，因此也就不会发现bug。这时他会为自己的盲目自信感到后悔，并重新扩大搜索范围。在猜数字的游戏中，如果你的“朋友”选择了135，你很快就会猜到100，若发现100仍然猜小了，于是你就会选择一个新的范围。

后面将会有更多内容介绍如何检验你的猜测，这也是拓宽搜索范围的一种方式。

6.1.2 你在哪一侧

在前面的旅店例子中，实际上技术人员在最开始的问题搜索中运气相当不错，因为信号流动的两个方向上都有问题，而且问题发生在线路的同一个地方。无论他查看哪里，都能够看到差的信号，他只需沿着差信号传来的方向移动即可。

然而，在第二部分的搜索中，只有发送到终端的数据发生错误，而来自终端的数据是正确的。这是大多数调试场景的典型情况：事情在开始的时候很正常，但中途的某个地方出错了。数据在系统中流动，当遇到bug时，数据流中断。程序在一段时间内运行良好，当它遇到bug时就发生了崩溃。你开车前进，挡风玻璃一直很干净，而当一只臭虫（bug）撞到玻璃上时，玻璃上就留下了一块污渍。

你必须知道搜索范围，而且必须知道在一端一切正常，而在另一端出现了问题。让我们把这两端分别称为上游（好的、干净的水）和下游（坏的、发出臭味的、粉红色的水）。你需要找出工厂的废水管，就是它把带有臭味的、粉红色废料排放到河里。每次你观察一个点，如果情况正常，则可以认为问题出在这个点的下游。如果水是粉红色的且发出臭味，则可认为问题发生在上游。

当问题来自一个排放污水的工厂时，问题很明显，但这与电子或软件有什么关联呢？通常，在硬件和数据流软件中，下游就是指信号或数据流的远端。如果问题是软件崩溃，则下游就是代码流的后面。（在这种特定的情况下，可以在某个位置设置一个断点或消息，如果

代码的执行能够到达这里，则崩溃就发生在这一点的下游。如果在到达这一点之前软件就崩溃了，则说明问题发生在上游。）如果一个复杂的软件计算发生了错误，则可以在计算过程的中间停止计算，查看到这里为止的计算是否正确。如果不正确，则向上游移动（更早），如果正确，则向下游移动（更晚）。

在我设计的那个称量金属粉末重量的系统中（参见第3章），中断信号是由天平发给计算机的，因此天平是上游，计算机是下游，而bug位于它们之间的控制芯片中。在第5章的视频压缩例子中，运动估计算法计算一个要搜索的新位置，然后尝试匹配图像。我们看到系统没有正确地搜索新位置，因此忽略了下游的匹配逻辑，而查看上游的搜索逻辑。

6.2　插入易于识别的模式

当清澈的水变成粉红色并发出臭味时，我们很容易发现。但是，当bug的效果很微小，或者数据看起来完全是随机的，因此甚至一个很明显的错误也无法找到明显的原因时，该怎么办呢？一种使得微小的效果变得更明显的方法是使用一个真正易于识别的输入或测试模式。用河流作为类比，正常数据就像是一条混浊的河流，废物很难识别。你必须清除淤泥，并使用干净的水来判断。在前面的"不要想，而要看"的案例故事中，我描述了高级工程师无法从随机的程序数据中看出问题，于是他把"00 55 AA FF"加入到"河流"中，因此很容易就看到了它是什么时候被"污染"的。在下一章开头的案例故事中，你将会看到一个测试模式在经过一系列步骤后，是如何帮助找到一个音频bug的。

当我使用视频引擎时，通常使用一种能够在屏幕上平滑改变颜色的视频模式，以便映射错误可以显示为行或边。如果你曾经在你的电脑中改变视频分辨率，那么就已经看到过这种技术了，当你点击"测试"按钮时，会给出一系列的模式和颜色，所有模式都已经标注了名称，因此你知道它们应该是什么样子的。如果一种视频模式显示不正确，说明它不适用于你的计算机。

在运动估计的案例故事中，我们创建了一个已知的挥手输入，我们知道这个挥手动作应该显示在屏幕上的哪个位置，也知道它的方向。我们可以查看计算数据，而且知道它的结果应该是什么。



案例故事 在上一章中，我描述了一种使用VCR来捕获视频输出的情形，并在输出中查找顺序有误的帧。我们之所以能够分辨帧是否发生了顺序错误，是因为我们的处理方法相当于把输入图像用不同颜色分隔成一个一个的帧（就像是一张五颜六色的比萨饼），然后每4秒钟播放一帧。每一帧（1/30秒）都有标记。这种分隔方法使得每播放一帧，就移动一个标记。因此，如果有哪个帧没有按顺序播放，或者有一个帧被重复播放了，很容易看得出来。

现在再讲述另一个视频案例，这次，我们需要查明音频与视频为什么没有完全同步——出现了“假唱”问题。我们构建了一个程序，它在发出滴嗒声的同时把屏幕的一块区域由白色变为黑色。当我们把它输入到音频–视频同步器中时，就很容易看到音频和视频到底是在哪里“对不上口形”。因为在视频流中声音和视频的大的改变变得非常明显，甚至在高度压缩之后也是很容易看出来。

在过去使用Motorola 6800微处理器的时代，指令码DD将会导致处理器无限循环，依次从每个内存地址读取数据。[有些工程师把这称为“Halt and Catch Fire”（HCF）指令，但我们把它叫做 “Drop Dead”，并因此记住了它。] 当用示波器查找硬件时序和地址逻辑问题时，Drop Dead模式非常有用，所有的地址和时钟线路都是整齐的、循环的方波。■

当然，在植入已知的输入模式时，注意不要因为设置了新的条件而改变bug。如果bug与模式密切相关，那么植入一个人工设置的模式可能会将问题隐藏起来。因此，在植入模式之后，应该在继续调试之前“制造失败”。

6.3 从有问题的支路开始查找问题

很多系统都有多个流程汇合到一起，这非常类似于支流汇入干流。如果从主源头开始搜索，可能会由于找错了支流而浪费大量时间。不要采取这种做法。不要从好的一端开始去确

认一些正确的事情，正确的事情太多了（这也是你所希望的）。从错误的一端开始（也就是从发出臭味的粉红色排放物开始），然后向上游追查。把分支点作为测试点，如果问题仍然在上游，则分别查看每个分支的一小段，以便确定哪个分支有问题。

假设你的炉子点不着火了。你可能猜测油用光了，于是检查油罐，发现它是满的。如果你想确认一下油的流动没有问题，需要沿着输油管检查，确认所有输油管都没有问题，并最后证实喷油嘴也能正常喷出油，这会浪费很多时间（而且你也会沾一手油）。但是，你很聪明，而且你阅读了本书，因此你从炉子这一端开始，很快就确认燃料是正常的，而电力有问题。你来到继电器盒这里，发现有几种可能的支路：总电源、自动调温器和防火断路器。你可以到总闸那里看一下总电源是否正常，但继电器盒上的电表告诉你电力正常，因此这条支路就不用考虑了。你可以拆下自动调温器来检查一下，但继电器盒上的仪表再次告诉你它对热量的反应很正常，因此这条支路也可以忽略。继电器盒上的仪表显示防火断路器出了问题，于是你决定去检查一下防火断路器。断路器就在你的头上方，它被固定在热风管道上了，这里是最热的地方，正好紧挨着炉子。于是你打电话请水管工人更换了一次性的保险丝，并把感应器移到别的地方，使它只有在真正起火时才熔断保险丝。

在运动估计的案例故事中，问题是输出视频的质量不高。这个问题有两个向上的分支，一是运动估计，二是图像编码。我们选择了运动估计这个方向，这是正确的。系统没有搜索到所有的运动。如果我们在所有运动方向上都能够看到许多彩色的小方块，这说明运动估计没有问题，因此我们就会去查看图像编码这条支路，而不是去检查运动估计。在很多情况下，我们不是从起点开始并验证图像编码，这将需要验证频域转换、行程编码、变长编码、量化以及十多种听上去非常复杂的软件领域，它们都属于图像编码这个分支。事实上，它们确实非常复杂，因此很难验证。此外，它们并没有出错，所以所有这些复杂的验证都是在浪费时间。

6.4 修复已知 bug

有时，我们很难相信一个系统中会有多个bug，就像在旅店预订的例子中一样。这使得

用“分而治之”原则隔离每个bug变得更加困难。因此，如果同时出现了多个问题，当你确实查明了其中的一个问题时，应该立即修复它，然后再查找其他问题。我总听人们说“那里出问题了，但它不可能影响我们正在查找的问题。”事实上，它确实（而且经常）会产生影响。如果你修复了已知的错误，就可以专心致志地查找其他问题。在旅店预订系统中，技术人员只有在修复了转接盒中的双向电阻阻值过高的问题之后，才能够发现速度最慢的那台终端的接线问题。

有时修复了一个问题，另一个问题也解决了，两个问题实际上是同一个bug。

此外，如果修复某个问题对其他的问题有影响，一定要首先修复它之后再测试其他的问题。如果修复了一个问题后将会引发新的问题，那么你可以尽早发现，并有更多时间处理新的问题。

6.5 首先消除噪声干扰

前一条规则的一个推论是，有些特定类型的bug可能会引起其他bug，因此应该首先查找并修复它们。在硬件中，噪声信号可能会引起各种难以查找的间歇性问题。在寻找问题之前，首先应该注意短时脉冲波形干扰和时钟的回声、模拟信号的噪声、时序波动以及电压不稳等干扰因素。其他的问题往往难以预计，而且当清除噪声后，它们就消失了。在软件中，差的多线程同步、意外的重入例程（reentrant routine）以及未初始化的变量会导致系统产生很多随机行为，从而为你的工作带来极大的麻烦。

但不要过于极端。如果你只怀疑噪声就是问题，或者时间问题很微小时，那么就要做一个权衡的考虑，看看修复问题的难度有多大，再看看它是否确实会引起问题。前面讲的那几位初级工程师通过制作一块新的电路板来解决时间问题，他们只怀疑问题出在时间上，而且他们的修复方法难度很大。这种修复只是耽误了真正的研究时间。此外，人们也很容易成为一个“完美主义者”，为了达到全面的高质量把你发现的所有不好的设计都“修复”一遍。你可能只是因为先前的程序员编写的GOTO语句看起来很差劲就删掉它们，但是，如果它们并没有实际引起问题，最好还是保留它们吧。

6.6 小结

分而治之

当bug的藏身之地不断被缩小一半时，它将很难再隐藏下去。

- **通过逐次逼近缩小搜索范围**。猜测1～100内的一个数字，只需7次。
- **确定范围**。如果数字是135而你却认为它在1～100内，那么你必须扩大范围。
- **确定你位于bug的哪一侧**。如果你所在的位置有排放物，则排放管就在上游。如果没有排放物，则排放管就在下游。
- **使用易于查看的测试模式**。从干净、清澈的水开始，以便当排放物进入河流中时很容易看到它。
- **从有问题的一端开始搜索**。如果你验证的是正确的部分，那么需要验证的地方太多了。应该从有问题的地方开始，然后向后追查原因。
- **修复已知bug**。**bug互相保护，互相隐藏**。因此一旦找到，立即修复它们。
- **首先消除噪声干扰**。注意那些导致系统问题的干扰因素。但对一些无足轻重的问题不要过于极端，也不要为了追求完美而去修改所有地方。

第 7 章

一次只改一个地方

"有人说天才就是无止境地吃苦耐劳的本领。这个定义下得很不恰当，但是在侦探工作上倒还适用。"

——福尔摩斯，《血字的研究》



案例故事　在一个周末，我们请一位调试高手来为我们的一位软件工程师帮忙。这位软件工程师在调试系统时遇到了困难，此系统用计算机来处理声音，通过音频处理硬件和软件"管道"（包括另一家公司的软件）加工处理，最后用一个扬声器输出声音。当然，输出的声音质量很差，要不然他们就会愉快地度周末去了。当数据在系统中传输时，在某些位置上数据被打包为块，并带有一些额外的"帧指示位"（framing bit），用于定义每个块的开始和数据类型。在其他时间，这些帧指示位就会被删除。系统的某些部分假定帧指示位存在，而其他部分则假定它不存在（参见图7-1）。软件工程师猜测在某些位置的帧指示位可能丢失了，于是改动了一个地方，加上了帧指示位。但声音质量仍然很差。

调试高手到达后，立即坚持说应该在数据处理流程中输入已知的数据，并利用插装工具来观察失败。他们花费了好长时间找到了一种合适的测试模式并找对了观察的位置，最后发现数据被破坏的地方，并由此追踪到了原因，原来是一个缓冲器指针错误。他们修复了这个bug，看到测试模式通过了先前的故障点而没有发生错误，于是他们满怀信心地输入了实际的音频数据。结果，声音仍然很糟糕。

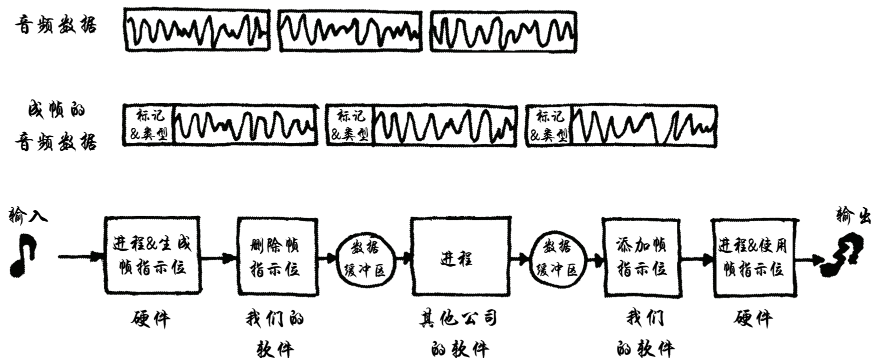


图7-1　声音失真的系统

他们有点摸不着头脑了，于是重新查看了数据流。或许他们没有正确修复bug？或许是修复没有生效？他们花费了一个小时重新确认了问题确实已被修复。然后又重新检查代码，确认修复代码确实运行了。在这个时候，软件工程师突然拍拍自己的额头，说："我先前为了添加帧指示位，把一个处理程序修改了，但这并没有解决任何问题，而我并没有把它改回来！"事实证明下游的处理器认为这个额外的帧指示位也是音频数据，因此也把它播放出来了，所以声音听上去很糟糕。他删除了先前的"修复"，系统工作得非常好。这位调试高手为我们引出了"一次只改一个地方"这条规则，并为本书提供了这个案例故事。■

我们的软件工程师为了修复问题而更改了一个地方，但这个修改并没有解决问题，而他认为这不会产生什么影响。这是一个非常错误的假设。它确实有影响，它使得音频数据发生错误，只是两次错误与一次错误也没有什么太大的区别。当原来的错误被修复后，他犯的错误仍然存在，因此声音的质量仍很差。当他的修改没有解决问题时，应该立即把它改回来。

假设有一天你想开你妻子的车去上班，但车子却无法发动。你注意到档位处于停车档（Park），你猜测需要把它换到空档（Neutral）才能发动，于是你换到空档然后再次尝试发动车子。仍然无法发动。你有点困窘地问你的妻子车子怎么了，她告诉你车钥匙不太灵敏，需要把它转到头才能打着火。你使劲地转动钥匙，几乎就要把它扭断了，可是仍然无法发动车

子。事实上，如果你在使劲拧钥匙之前把档位从空档换回停车档，车子就会发动了。

7.1 使用步枪，而不要用散弹枪

一次只改一个地方。你一定听说过“散弹枪方法”（指全面撒网），忘掉它吧。找一支好的步枪，你将会更好地修复bug。我知道有些技术人员在修理坏的电路板时只是换元件，他们可能一次换掉三四个元件，有时发现问题解决了。这种方法很省事，但他们并不知道哪个元件坏了。更严重的是，这样盲目更换元件有可能破坏其他正常的元件。

此外，如果你*真的看到*了错误，应该只修复这个地方。换言之，如果你认为你需要一支散弹枪来击中靶子，而问题却在于你无法看清靶子。这时，你真正需要的是更好的光线和一副新眼镜。

社会科学家和医学研究人员通常利用兄弟、姐妹（有时是双胞胎）来分离他们正在研究的因素。他们可能观察一些分开生活的双胞胎，由于双胞胎的基因是完全相同的，因此任何差别都来自环境因素。同理，他们可能观察被领养的孩子，把家庭环境作为常量，并假设任何差别都来自基因。当牙医寻找你口腔中对冷物过敏的牙齿时，他会使用一种类似于步枪的仪器喷出冷空气流，一次检查一颗牙齿，而不会在你的嘴里放一个冰块。过去，人们会用一长串灯泡来装饰圣诞树，如果有一只灯泡坏了，整串灯泡都不会亮。这时，你一次只会更换一只灯泡，直到整串灯泡重新亮起来。如果你把所有灯泡都换掉，将会花很多钱，而换下来的灯泡中只有一个是坏的，其他都是好的。

这就是科学方法。为了看到一个变量的影响，科学家尝试控制可能影响结果的所有其他变量。在小学中，有一种流行的植物生长实验，你用的是完全相同的土壤、相同的灌溉方案和相同的种子，而改变日照时间。因此，植株大小的任何差别都是由日照时间造成的。如果使日照时间完全相同，而改变灌溉量，则任何差别都来自灌溉量。如果你在解决植株个头小的问题，同时改变了浇水用的水壶的颜色、灌溉方案和日照时间，那么你就不会知道问题是否与水壶的颜色有关。

如果你正在开发一个按揭计算程序，它在计算贷款时发生错误，那么可以把贷款额和周期固定下来，然后改变利率，看看程序的计算是否正确。如果正确，则把周期和利率固定下来，改变贷款额。如果这样也能正确计算，则只改变周期。这样，你就会发现计算中的问题，或者发现更奇怪的事情——你的奔腾处理器像是发生了算术错误（“但那是不可能发生的”）。这种奇怪的错误是由更复杂的bug导致的，而处理器正是使它们变得复杂的原因。隔离和控制变量的方法类似于把已知数据输入到系统中。它能够帮助你看到奇怪的事情是如何发生的。



案例故事　我正在为一个第三方设备编写软件，它从笔记本的VGA输出采集视频，以便把视频传输到另一台计算机并播放。由于笔记本的VGA信号有着各式各样的分辨率和时序，而且没有时钟信号告诉我们正在发生什么，因此不得不对采集到的视频画面进行采样，并通过观察视频画面露出的边缘（黑边）来判断图像有多少像素、时钟的速度以及每个像素的边缘在哪里（以采集卡的时钟为参照）。

这项工作很难，因为每次测量都需要对时间进行猜测，然后根据猜测来测量结果，接着再使用这些结果来计算为下一个参数猜测的时间。当然，我首先试着运行这个系统，不幸失败了（人们总是要自己试一下才相信，不是吗）。每次测量都会影响下一次测量的时间，因此很难分析结果，但问题的关键看起来是确定像素的边缘在哪里（以采集卡的时钟作为参照）。

我再次运行了我的步骤，但忽略了所有的测量，除了像素采样点延迟（相位）参数以外，所有参数均设置为默认值。这次我完全手工移动所有的8个位置（参见图7-2）。当采样点经过一个输入像素到达下一个像素时，系统应该采集前一个时钟的视频画面。我希望看到的是输出视频向左跳一个像素，但相反，它却向右跳一个相位，然后在后一个相位再次向左跳回一个相位。这种跳跃是毫无意义的，但由于其他的变量都已经固定不变了，因此我知道错误就在这里。

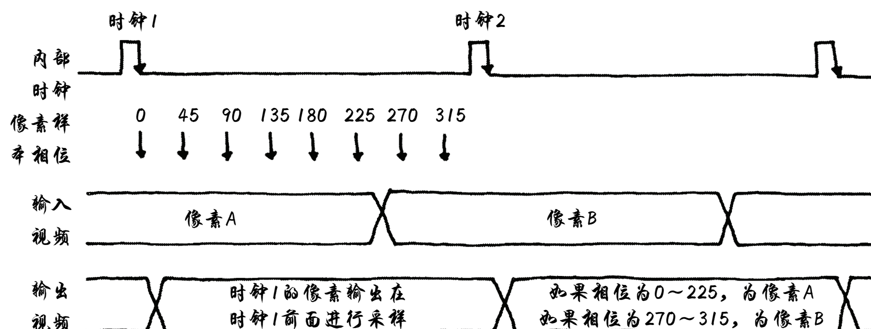


图7-2 找到像素的边缘

我把数据拿给第三方供应商看，供应商检查了问题，发现相位参数的说明文档有错误。当我把参数从0增加到359时，实际的采样相位并没有像文档说明的那样从0°增加到359°，而是从0°增加到89°，然后跳回到−270°，接着再向前跳到−1°。当相位向回跳时，视频就向右跳，然后当它重新穿过像素边缘时，又向左跳回到起始位置。■

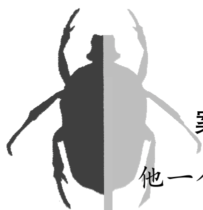
如果我没有把其他变量都固定下来而只改变相位（按照已知的值来改变），那么我永远也不会想到是相位参数出了问题。由于相位是唯一改变的参数，因此它必然是跳帧的原因。

7.2 用双手抓住黄铜杆

在很多情况下，你可能想改变系统的不同部分，以便看看它们是否对问题有影响。这往往是一个危险的信号，说明你正在猜测，而不是使用插装工具来观察正在发生什么。你正在改变条件，而不是捕捉错误的自然发生。这可能会把最初的错误隐藏起来，而且引起更多错误。前面例子中的那位软件工程师添加音频帧指示位的做法正是犯了这种错误。

在核潜艇中，动力装置控制台前面有一个黄铜杆。当发生某种状况而启动警报时，工程师们必须双手抓住黄铜杆，并一直保持，直到看清所有仪表和指示器，明白发生了什么事情。这样做是为了帮助他们克服开始“修复”问题的冲动（急着按开关和打开阀门）。这些快速

的修复举动会扰乱系统的自动修复功能，由于快速设置新的条件而隐藏了原来的问题，而且还有可能引发真正的大灾难。更有效的方法是记住要做的事情（“抓住黄铜杆”），而不是记住不要去做什么事情（“不要动仪表盘”）。因此要抓住黄铜杆！



案例故事　有一年，我们的宿舍举行了一个圣诞party，一位室友（我们送了他一个绰号“笨笨”，其实这个绰号很适合他）把他的立体声音响拿到了起居室里，想在节日放点音乐。他沿着墙布置右边扬声器的线，当到了壁炉时（先前一直没人使用壁炉，虽然新到的木柴已摆放好，随时可以生火），他把线布到了圆木的后面。当然，晚上，有人点起了壁炉，在圣诞火焰的烘烤下，右边的扬声器很快就不出声了。导线上的绝缘层被烤化了，扩音器右边声道的线路发生短路，烧断了右声道的保险丝。大家谁也没有想到这一点，而且这时他们正忙着跳舞，“蛋酒”[①]的作用也把他们搞得晕头转向，因此他们并不打算中断音乐或查看一下保险丝。他们决定看看问题是否出在扬声器或扩音器上，于是他们把扩音器的左右扬声器的线对调了一下。当他们重新打开扩音器时，左边声道的线也短路了，左边的保险丝也被烧断。现在两个扬声器都哑了，他们只好在没有音乐的伴奏下度过了这个party。如果他们不做任何事情（用双手拿着盛装蛋酒的酒杯），情况就会好多了。■

7.3　一次只改变一个测试

有时，改变测试序列或一些操作参数可以使问题更加有规律地出现，这有助于观察错误，而且可能会帮助我们找到问题的线索。但我们仍然应该一次只改变一个地方，以便判断哪个参数有影响。如果做了一个改变后看上去没有什么效果，应立即把它改回来。

① 蛋酒，eggnog，是圣诞节最具代表性的饮品，由鸡蛋、牛奶和朗姆酒调制而成。

7.4　与正常系统进行比较

一旦你掌握了某种可以制造失败的方法（即使只是随机出现），那么你就有一个绝佳的机会成为一名出类拔萃的工程师（做一名出类拔萃的工程师！放眼全世界！或至少看到你的特异之处）！使用两个例子，一个失败的例子，一个正常的例子，对比它们的示波器观察结果、代码、调试输出、状态窗口或你插装的任何工具所显示的结果。我曾经用这种方法找到很多bug，同时运行系统的一个正常的例子和有问题的例子，然后并排观察两个调试记录，注意它们之间的区别。“看，正常的视频电话呼叫的电话号码字段设置是1-700-VID-TEST，而出错呼叫的设置是1-700-BAD-JEST。”

如果你在两个测试之间更改了很多代码，或者为两个测试设置了不同的环境，那么这两个测试将很难对比。它们之间有很多差别并不是由bug引起的，而你必须不断地解释这些差别。你必须把它们之间的差别减少到只与bug有关。排除其他的干扰因素。试着从相同机器的连续测试中获取调试记录，不要使用不同的机器、软件、参数、用户输入，也不要在不同的时间和不同的环境下进行测试。甚至不要穿不同的衬衫，它可能会把bug隐藏起来。（不要笑，下一章将会讲述一个案例故事，其中我的衬衫就是一个关键因素。）

这并不是说你不应该对那些与bug无关的方面进行测试。实际上，你并不真正知道什么与bug有关，因此需要对一切能够测试的地方进行测试。如果通过测试发现某个方面与bug无关，则可以把它从两个调试日志中删除。是的，你需要对大量的无关数据进行过滤，但相同的无关数据在两个调试日志中都会出现，因此可以忽略它。在第4章中，我们曾讨论过，通过查看我们所捕获的正确视频呼叫日志和错误呼叫日志之间的差别来寻找问题，我们并不知道要找什么，因此不得不过滤掉大量的类似数据，最后看到了错误呼叫中的非法命令。

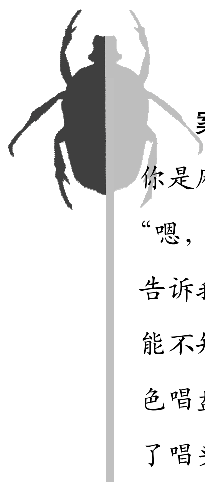
但是，这并不像听上去那么容易。经常有人请我解释我是如何做的——通过查看几组日志就找到了问题。还有人请我为测试人员讲授一个入门课程，教给他们如何这样做。也有人请我编写一个软件过滤器，用于读取一个日志，并自动找到出错的数据。然而，有关如何查找错误数据的知识并不适合教给初学者，这也不是通过编写一个程序就能做到的。你要查找

的错误绝对不会和上次一样。要想过滤掉那些由时序或其他因素引起的差别，需要相当丰富的知识和智慧。这些知识超出了初学者所能达到的水平，这种智慧也不是软件能具有的（还记得“人工智能”是怎样淡出人们的视线的吗）。软件所能做的贡献也就是帮助我们对日志进行格式化和过滤，从而当你用具有超级智慧的人类大脑（你有一个充满智慧的大脑，不是吗）来分析日志时，能够快速找到差别（或者是导致差别的原因）。

当你查看冗长的、复杂的日志时，你可能很想只查看可疑的部分，如果你有了线索，那么这是个不错的想法。但如果你没有线索，就准备好查看整个日志吧，因为你并不知道区别在哪里。

有一点必须提醒你，这可能是你以前未曾做过的最枯燥的任务。泡一杯浓咖啡，用一根牙签把眼睛支上，一点一点查看调试日志吧。（仔细想想，还是不要这样做了，这会使你的眼珠子掉出来[①]！）

7.5　自从上一次能够正常工作以来你更改了什么



案例故事　我上大学时，曾经在一个家具厂打工，有一天一位工友问我：“嗨，你是麻省理工学院的，可能非常了解立体声音响，或许你能帮我个忙。”我回答说：“嗯，我对立体声音响也不是完全精通，但我用过它们，我想或许能帮上忙。”她告诉我说，她刚刚把唱机转盘拿去修了，现在它听上去很糟糕。（有些年轻读者可能不知道唱机转盘是什么，它就是电唱机，播放那种很大的用聚乙烯材料做的黑色唱盘，唱盘买来的时候装在纸制护套里，盘的两面都有音乐。）她说修理店更换了唱头（唱头从唱针生成电信号，唱针则放在唱片上）。我们来到了她的工位，她

① 原文：You could put an eye out with those things.这是美国习惯用语，家长常常用这句话来警告孩子，意思是不要做危险的事情。

放了一张唱片，声音确实糟糕极了。事实上，这种声音让我想起了以前的经验：当我把便捷式录音机接到扩音器上时，如果把录音机的音量调得过大，扩音器就无法处理这么大的输入信号，于是声音听上去就很粗糙和失真。因此，我首先想到的问题就是“输音量过大”。

我知道（她告诉了我）自从上次唱机正常工作以来，有一个地方被改变了，那就是更换了唱头。我还通过安装立体声音响的经验知道有两种唱头，一种是磁性的，一种是陶瓷的，它们通常有两种不同的输入类型（在扩音器的后面，参见图7-3）。这是因为其中一个的声音大于另一个。这样，问题就很清楚了，她把低音量的唱头换成了高音量的唱头，但新的唱头仍然插到了原来低音量唱头的位置，现在输入的音量过高了。

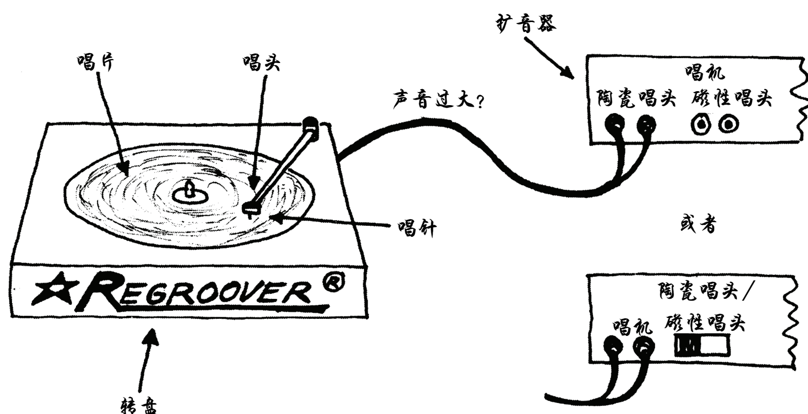


图7-3　老式唱机的工作原理

我并不知道现在使用的是哪种唱头，但我断定更换它的接线位置就会解决问题。我看了音响的后面，令我高兴的是，有一个切换开关，这比我想的还要简单。我把开关扳到另一侧，音乐立刻变得正常了。事实上，由于刚刚更换了新的唱针和唱头，音效比原来还要好。总共花费的分析时间：约30秒。所体现出来的总的技术能力：很强。（赢得的总的爱情数：零。在那个还没有互联网的时代，超强的技术能力还不是特别有吸引力。）■

有时，正常的系统和错误的系统之间的区别是由于一项更改造成的。做了更改之后，正常的系统开始出现故障。一种非常有效的办法是找出第一个导致系统出错的版本，尽管这可能需要连续测试原来的版本，直到找到没有故障的版本。一旦找到了这个版本，再前进到下一个版本，验证故障是否再次出现。做完这一步之后，至少可以把问题的范围限定到两个版本之间所做的修改。当然，你得有一个完备的源设计跟踪系统，这样就可以快速查看所有任何两个版本之间的所有区别。（如果你还没有这样的跟踪系统，现在就去获取一个。有关更多细节，参见下一条规则“保持审计跟踪”。）假设你所做的修改不是对系统进行全面的大幅度修改，这种方法将使你更容易找到问题。

通常，新的设计会出问题，这也是我们为什么总是在发布新产品之前对新设计进行测试的原因。有时一个部分的新设计与另一个正常工作的部分不兼容。前面所讲的立体声系统就是这种情况，新的唱头没有问题，只是需要改变扩音器的设置来适应它。

然而，有些情况较为复杂。有时问题已经存在了很长时间，但只是某个地方（例如时序或数据库大小）被改变之后，它才显露出来。你可能认为问题是在5.0版本时出现的，但实际上你所做的更改只是把问题暴露出来了，而问题自从3.1版本就已经存在。通常，一段新的代码或新的硬件修订设置了新的条件，结果使得原来一直很可靠的子系统出了问题。子系统有一个漏洞，只是你以前从未遇到它。你可能试图追踪由那个漏洞引起的bug，而有时这样只能暂时修复问题，而你实际需要做的是解决那个漏洞。



案例故事　在一所老房子里住了几年之后，在隆冬的一次暴风雨后我们发现二楼楼梯附近的天花板往下滴水。我走到楼上，试图找到漏雨的源头，最后来到了阁楼用于布线的线槽附近，水就是在这里从冰冻的屋顶上流进来的。水沿着一根斜梁流进，滴在一个被嵌入到斜梁中的小金属片上（大小和信用卡差不多）。水滴遇到这个斜插的小金属片后，滴到了地板上，再从地板淌进屋子里。旁边有一个旧的塑料盆，就像我们平时在水池里用来泡碗碟的那种盆。当我们刚搬进来时，

> 这个盆正好就放在小金属片的下方，但我在夏天布线的时候在这个地方爬来爬去，把它踢到了一边，因为我不知道它是干什么用的。
>
> 我在布线的时候，引入了一个“bug”：我把盆移了位置。当然，我采用了明显的短期解决办法——又把盆移回了原处。但这只是暂时的。第二年夏天，我们彻底修理了这个由于塑料盆移位所暴露出来的bug：重新修缮屋顶。■

7.6 小结

一次只改一个地方

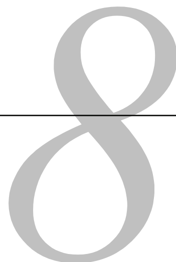
我们在生活中要有一点先见之明。如果你所做的更改没有起到预期的作用，那么就把它改回来。它们可能会产生无法预料的影响。

- **隔离关键因素**。如果你在检查日照时间的影响，就不要改变灌溉方案。
- **用双手抓住黄铜杆**。如果你在不知道具体发生了什么问题的情况下就试图去修理核潜艇，可能会引发一次水下的切尔诺贝利爆炸[①]。
- **一次只改一个测试**。我之所以知道我的VGA 采集相位被破坏了，就是因为其他东西都没有发生改变。
- **与正常情况进行比较**。如果所有出错的情况都有一些特征，而这些特征是正常情况所没有的，那么你就找到了问题所在。
- **确定自从上一次正常工作以来你改变了什么地方**。我的工友改变了唱机转盘上的唱头，因此这是一个很好的调试起点。

① 1986年苏联时期乌克兰境内的第一座核电站反应堆爆炸，引发严重事故。

第 8 章

保持审计跟踪



“在侦探学的所有分支中，没有比足迹学这门艺术更重要而又最易被人忽视的了。”

——福尔摩斯，《血字的研究》



案例故事 我们正在调试一个视频压缩芯片，它用于传送视频会议信号，首先生成流畅的运动视频，并压缩成很少的字节，再通过电话线来传送视频。流畅的运动需要每秒钟有很多帧（或图像），我们把它设置为每秒钟30帧。虽然使用了一个很早的原型，但我发现帧率偶尔会突然从30下降到2左右，而且没有明显的原因。除非重启芯片，否则不会再快起来。新的芯片显然有一个bug，但令我不解的是到底是什么触发了帧率的下降。很快，我就肯定问题与芯片的运行时间长短无关。有时，它立即就失败了，有时它可以运行2个小时。而当我第二天再来时，系统完全正常了，没有发生一次错误。我想可能是由于房间温度变化的关系，于是试着对芯片进行加热和制冷，但没有效果。第二天芯片又开始出错。我想，这可能会使我疯掉（也许真的会这样），幸好有一次我注意到当我从椅子上站起来时，芯片突然失败了。我坐下来，重启处理器，看到它快速运行，然后我又从椅子上站起来，它再次失败了。或许它太孤单了，不想让我离开？

我意识到当我站起来时，我的衬衫会有更大面积进入到摄像机的镜头内。现在，由于我住在新罕布什尔州，我穿着当地生产的一种法兰绒格子衬衫（我经常穿这

一款衬衫)。而前一天，我穿的是一件纯蓝色的衬衫。前天，我穿的是另一件法兰绒格子衬衫。我做了几次实验，发现当视频压缩器尝试处理极难压缩的图案（活动着的格子衬衫）时，它就会停止工作（参见图8-1)。

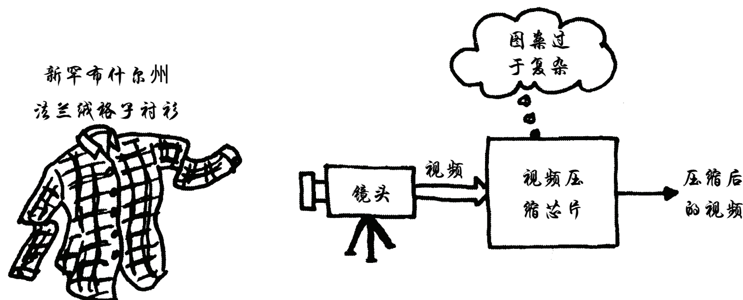


图8-1　视频压缩器与新罕布什尔州生产的法兰绒格子衬衫

我的同事和视频压缩芯片供应商很难相信衬衫会导致这种后果，但这确实很容易证实（事实上，我经常使用这些格子衬衫来演示衣服也可以令视频压缩引擎达到极限，在调试时也可以顺便为这款衬衫做做推销。)

我承认我没有记下我每天穿的是什么衬衫，因此我在测试的时候没有书面的审计跟踪记录。我并没有记录我从椅子上站起来这个动作是导致处理器失败的原因。但我记得这些事情，它们很容易记住。但当我把报告递交给芯片供应商时，我确实写明了格子衬衫引起的问题：穿着格子衬衫在摄像机前站起来时会导致芯片失败，而且芯片只有重启才能恢复。我甚至把衬衫的图案影印了一份，以便供应商可以再现问题（实际上，供应商想让我把衬衫脱下来给他们，但我没舍得)。想象一下我提供的这些信息会多么有用，如果我只是写明“芯片偶尔会减慢速度，需要重启才能恢复”或者只是写明“芯片偶尔会失败”，就不会有好结果。(实际上，我曾接到过这样的bug报告，只是简单地写了“它出问题了”。) ■

这个故事的要点是，有时看起来最不起眼的事情实际上却是导致发生bug的关键。在测试人员看来不重要的细节（格子衬衫）可能对于bug修复人员很重要。而在测试人员看来很明显的事情（芯片需要重启）可能在修复者看来无关紧要。因此，你必须记录下每一件事情，

不起眼的事情可能会很重要。

8.1 记下你的每步操作、顺序和结果

保持审记跟踪。在检查某问题时，要记下你所做的事、做事的顺序，以及发生的结果。每次都要完成这些记录。你是在检测测试步骤，就像检测软硬件一样。必须清楚每一个步骤和每步执行的结果，以此确定在调试时应重点关注哪一步。



案例故事 一位客户总是拨打客户支持电话，说他的软盘只能使用一次，然后就失效了。客户支持部门不断给他寄去新的软盘，但每张盘都只能使用一次。最后，客户支持人员决定获取一个实时的审计跟踪记录，他们让客户在电话中报告他的一步一步操作过程。像预期那样，软件第一次工作正常，然后，客户把它放到了一边——他用一块磁铁把软件吸在了文件柜上。■

当你发生食物过敏时，过敏症专科医师会问你吃了什么，试着把食物和你的反应关联起来，以便确定哪种食物引起过敏。如果食物和你的反应之间的关联不明显，医生就需要更多细节，他们可能会让你把吃的每样食物和每个反应记下来。这就是一种审计跟踪记录，直观且简单。它帮助医生发现吃草莓与荨麻疹之间的关联。（解决办法很简单，不吃草莓即可。）



案例故事 我过去每到星期日就会头痛，于是我在脑中回忆我的“审计跟踪记录”，最后确定原因是我在星期六没有像平时那样喝咖啡。头痛是由于体内咖啡浓度过低造成的。（解决办法很明显：买一台咖啡机，在星期六的早上做一杯双料的卡布奇诺咖啡。）■

当你去看心理医生时，医生会尝试提取你生活的“审计跟踪记录”（发生了什么事情，你对这些事有何感想），以便查明你是否有心理障碍。想象一下，如果你能把浓缩的详细简历提供给医生，将会节省多少诊费！（我知道，这是不现实的。事实上，对一位精神病患者进行调试几乎是不可能的，因为你的“插装工具”不起作用，他会遗忘、隐瞒和撒谎，你无法进入他的身体修复问题，而只能靠谈话来引导他进行自我修复。）

8.2 魔鬼隐藏在细节中

遗憾的是，虽然审计跟踪的价值已经被普遍认可，但所需的详细程度却没有被接受，因为很多信息都被忽略了。正在运行的系统是什么类型的？导致失败的事件序列是什么？有时甚至连具体是什么故障也没有说清楚。有时报告只是说“它出错了”，但并没有说明图形是完全混乱了，还是所有红色区域都变成了绿色，或是第三个数字发生错误。报告只是说发生了错误。

更糟的情况是，为了获取细节，我们让报告bug的人把失败的调试日志记录下来。于是，我们现在有了一份bug报告和3个日志。他们说明哪个是故障日志了吗？没有。他们是否说明了问题的症状？没有。他们会说“全部都在日志中”。虽然日志中记录了所使用的测试方法，但测试人员所看到的故障细节并不在日志中。

假设你记录你的食物过敏日志，但你只记下了吃过的食物，而没有记录你突发荨麻疹，那么医生也将束手无策。关键是在记录调试跟踪或日志的同时，也要把那些没在日志中出现的所有条件和症状记录下来。如果你能把症状和时间戳记下来就更好了，参见下一节。

描述事情的时候要具体且一致。在视频通信中，我们经常有系统A和B，有时甚至有C，它们全部都尝试互相通信。我们会得到一个bug报告“没有远程视频”。那么，到底这是指远程端的系统A还是B？此外，是远程端不显示视频，还是没有从远程端发送来的视频？只有掌握了基本的症状后，才能开始调试问题。

除了记录发生了什么事情以外，另一个需要注意的细节是问题的严重程度。例如，在视

频会议系统中，在建立连接和断开连接的时候，不同供应商设备之间的交互可能会产生少量噪声。这时，需要注意的一个重要问题是噪声持续多久，以及它的干扰性有多大。只持续半秒钟的“嗡嗡”声完全可以忽略不计，而持续6秒钟的尖锐的噪声可能需要仔细研究一下。

如果你上过高中的化学课，可能会记得那个经典的“描述一支蜡烛”的实验。你必须写出蜡烛的50个不同的特征。在你冥思苦想一会儿后，老师会给出一些提示，其中有一条重要的提示是“为读者提供足够多的信息，让他们能够准确地理解你的体会”。细节！仅仅说明在你点燃烛芯后蜡烛会发光发热是不够的。发出的热和光的量有多大？如果有人要复制你的实验，那么在他“引爆”（点燃）蜡烛后，是否需要躲到掩体后面？它发出的热量使你无法把手平放在火焰上方6英寸的地方，只需2秒，你就会把手拿开。当然，掩体是不需要的。



案例故事　我的一位老相识正在开发一个硬件项目。他无意间碰到了电源箱，感觉到了一股弱电流（换句话说，他被温和地电击了一下）。硬件产品电到人是很不好的，这就像是被火烫了一下，于是他打算仔细检查一下。但他并不是十分肯定他感觉到的是真实的电击，还是正常的声波振动。因此他拉来一位同事，想让他触摸一下电源箱，以便确认一下。那位同事什么也没有感觉到。我的朋友又试了一次，仍然感觉到了电击。他又找来几位同事确认，所有人都没有感到电流。而他仍然在不断尝试，同事们围在他周围，都把手背到身后，看样子准备要把他送到疯人院了，这时他们注意到他没有穿鞋，而他们都穿了鞋。鞋子的绝缘作用足以使他们不会被电击。虽然我的朋友可能由于赤脚在硬件实验室中工作而受到批评，但他发现的bug并不是幻觉。■

就像那些只喜欢吃香草和巧克力冰淇淋的驾车者一样，有些关键的细节你可能从来都没怀疑过。不过既然你已经阅读了本书，那么就从现在开始怀疑并注意所有的细节吧。

8.3　关联

将某些症状与其他症状或调试信息关联起来是非常有用的。“线路刚刚接通时它会发出很大的噪声”比“它发出很大的噪声”要好。但最好的描述是“它发出很大的噪声，从14:05:23开始，持续4秒”。利用这条信息，当我查看调试日志时，就会查看14:05:23到14:05:27之间的几条音频控制命令。我可以非常肯定这些命令与问题有关。

再次回到食物过敏日志的例子中，如果你在一张纸上记下了你吃的食物和时间，而在另一张纸上记录你突发了荨麻疹，但并没有记下荨麻疹的发生时间，那么医生还是束手无策。

在有多个设备进行通信的系统中，应该把两个系统的时间调整为同步并跟踪它们。这样得到的跟踪记录将会是非常有用的信息。当然，在时间不同步的情况下进行分析也不是不可能，但你需要查看来自两台互相通信的不同机器的日志，并做一下心算——把其中一个日志的时间减去1分钟零23秒，因为它的时钟比另一台机器快这么多。然而，这种方法增加了难度而且令人厌烦。因此，不妨花点时间把它们调成相同的时间。

最后一次回顾食物过敏的例子。假设你以伦敦时间记下了所吃的食物，而以旧金山时间记下症状。医生不会束手无策，但会和你一样感到不耐烦。

很多bug都是通过把症状和人员时间表关联起来后发现的。



案例故事　有一个乱码字符问题被确定与Fred有关。Fred是一个大腹便便的人，他站起来拿咖啡壶时，肚子就会压到键盘上。

在另一个故事中，有一个崩溃问题被认定与George有关。崩溃的原因是文本缓冲区溢出。George自己“发明”了一种输入方式，只是这种方式会把同一行字符输入命令字段两次。他在电传打字机键盘上输入一行字符，但在到达右端之前，他会用手抓住打印头把它推回左端，而程序是依靠机械的自动回车来限制输入的。

> 第4章描述的计算机中心崩溃问题与下午3点休息期间自动售货机的大量操作有关。■

8.4　用于设计的审计跟踪在测试中也非常有用

第7章提到过源代码控制系统。它们是程序和工具文件的数据库，你可以利用它们来重建任何先前的软件版本（在已创建了新版本之后）。当很多工程师共同开发一个项目时，这些系统可以避免他们各自的代码修改互相干扰。（遗憾的是，它们不能把那些正确的代码整合到一起。）它们还提供了设计的审计记录，以便你能够知道系统在什么时候做了什么更改，并且在必要的时候可以恢复到一个已知的状态。这对于设计过程是很有利的，但对于调试过程也有用。当系统的某个版本显示出bug时，你可以有一个变更记录，记下了系统自从上一次正常工作以来都做了哪些修改。如果其中的条件都相同，那么你就可以准确地知道哪些代码修改引起了问题，并从这些地方入手来解决问题。.

源代码控制系统现在又称为“配置控制系统”，因为它们不仅仅跟踪程序代码，还跟踪你用于构建程序的工具。工具控制对于准确地重建版本是至关重要的，你应该确保有一个这样的控制系统。如后文所讨论的，如果有些工具变化你没有注意到，可能会导致一些非常奇怪的问题。

8.5　好记性不如烂笔头

在细节方面，永远都不要相信你的记忆，而要把它写下来。如果你相信你的记忆，将会制造很多麻烦。你会忘掉一些你认为不重要的细节，当然，这些细节将会被证明是非常重要的。你会忘掉一些在你看来不重要的细节，而这些细节对于后来解决另一个不同问题的人可能很重要。除了口头表述以外，你无法将信息传递给别人，而这会浪费所有人的时间。你无法准确地记住事情是如何发生的、发生的顺序以及事件之间有何关联，所有这些都是非常重要的信息。

把事情记下来。最好用计算机来记录，这样可以进行备份，并把它附加到bug报告后面，这样就很容易发送给其他人，甚至可以用自动分析工具来过滤它。把你做的事情和结果记录下来。保存调试日志和跟踪记录，并且注明相关的事件和影响（日志本身不会记录这些内容）。把你的推理和修复操作以及其他内容全部记录下来。

国王说："那个恐怖的时刻，我永远，永远也不会忘记。""你会的，"王后回答说，"如果你不记一个备忘录的话。"

——刘易斯·卡洛尔

《镜中世界》

8.6 小结

保持审计跟踪

不要只是在心里记住"保持审计跟踪"这条规则，而要把它写下来。

- **把你的操作、操作的顺序和结果全部记录下来。**你上一次喝咖啡是什么时候？你的头痛是从什么时候开始的？
- **要知道，任何细节都可能是重要的。**视频压缩芯片的崩溃是由于格子衬衫造成的。
- **把事件关联到一起。**"它发出噪声，从21:04:53开始，持续4秒"比仅仅说"它发出噪声"要好得多。
- **用于设计的审计跟踪在测试中也非常有用。**软件配置控制工具可以告诉你哪次修订引入了bug。
- **把事情记录下来！**无论那个时刻多么恐怖，都要把它记到备忘录中，这样你才不会忘记。

第9章

检查插头

9

“没有什么比一个显而易见的事实更能迷惑人了。”

——福尔摩斯，《博斯科姆比溪谷秘案》



案例故事 1984年6月，我们全家搬进了一所建了90年的老房子里，我们很快就熟悉了房子里的各种系统，并使它们运转起来。看起来所有东西都是成双成对的，房子有两套供电系统（接线方式很奇怪，后面将会讨论这一点），有两个炉子用来驱动压力热水供暖系统，一个炉子烧柴，原来的房主把它作为主供热系统，另一个炉子烧油，他用作备用系统，而我决定把它改为主供热系统（有人说烧柴的炉子要费三遍事儿，劈柴、堆柴和烧柴。我要做的最大改动就是把自动调温装置的仪表从柴炉换到油炉上。）这所房子甚至有两套污水系统和两口井。

热水器是一个小的热交换器，它把来自主热水系统的热量传递给水箱和淋浴喷头的热水管（参见图9-1）。不幸的是我们必须整个夏天都烧炉子，但这并不是困扰我的问题。淋浴喷头出来的水不热才是问题。

看起来房间内每个用水的地方都会使热水的压力减小，我在洗澡时会起一身鸡皮疙瘩。我想买一个能够保持水温恒定的自动调温阀，但它太昂贵了。一位机械工程师朋友建议我只需一个压力平衡阀，但我发现已经有这个阀了。我想可能只是没有足够多的热水。

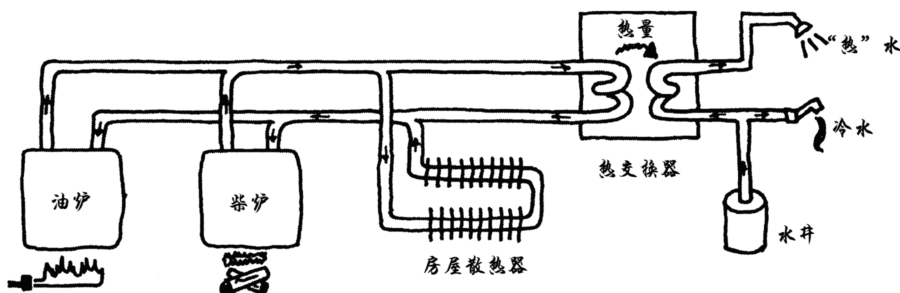


图9-1 独特的加热和热水系统

供热系统是“瞬时”的，这意味着水是随用随加热，而不是把热水存储在水箱里，这样不会出现热水用完的情况。因此，我猜测水温设置得不够高，但事实上热交换器的自动调温器被设置为140℉[①]，这个温度足够了。热交换器上的温度计显示水温也确实是140℉，一旦水温低于140℉，加热炉系统的热水泵就会启动。问题是当水流向淋浴喷头的时候，水温就无法保持了，而且水温降低之后，要想恢复为原来的温度，需要很长时间。我想这种瞬时加热水的想法并不够好，于是我考虑换一种系统。

当时已经是深秋了，在新英格兰，天气已经很凉了。我们发现供热系统在早晨的时候不能使房间迅速暖和起来。当然，压力热水供暖系统没有压力热空气供暖系统的速度快，但这个房子的供暖系统真是太慢了。当加热管工作的时候，我摸了一下它的温度，并不是很热。因此，我到楼下去查看了一下燃油炉的自动调温器，发现它被设置为165℉[②]。

正常的压力热水供暖系统应该设置为190℉[③]，我正在使用的是一个“压力温水供暖系统”。我发现原来的房主把烧柴的炉子设置为190℃，而把燃油炉设置为165℃，当然燃油炉只有在柴炉停止工作时才会启动。把备用炉设置为这个温度是没有问题的，但这个温度对于主系统来说太低了。我把炉子调到190℃，房间的供

① 约等于60℃。
② 约等于73.9℃。
③ 约等于87.8℃。

> 暖恢复正常了。
>
> 不仅如此，热水器也开始正常工作了。热交换器现在从190℉的热源吸收热量，而不是165℉。你知道，根据热力学定律，热交换器无法用165℉的热源把水加热到140℉，至少无法快速、长时间地给淋浴喷头供应热水。■

原来，一直困扰着我，令我在喷头下瑟瑟发抖的原因是我错误地假设热交换器的热源不会有问题。用“分而治之”的规则来解释，我把范围收得太窄了，因此错过了实际的问题。这种情况更可能发生在我所说的那些“一般性”或“基础性”元素上。由于它们都是一些基本需求（电力供应、热源、时钟），因此当你对一些细节进行调试的时候往往会忽略它们。

当然，压力热水供暖系统是一个我们不熟悉的环境，但这样的奇怪环境有很多，如果你忽略了基础性问题的可能性，那么有时肯定会遇到一些非常尴尬的场面，比如淋浴喷头的水太凉。

9.1　怀疑自己的假设

永远不要相信自己的假设，特别是当这些假设在一些无法解释的问题中是核心因素的时候。应该问自己一个古老的、看似愚蠢的问题：“插头插上了吗？”虽然这个问题看上去很愚蠢，但它经常发生。你可能费尽周折检查调制解调器软件为什么不工作了，事实证明你只是把电话线踢掉了。还记得我的那位使用水箱内置电热水器的朋友吗？他并没有想到是断路器的问题，而在水箱下面做了一下午的无用功。

通常，问题发生在较低的层次上。你可能奇怪为什么一个复杂的数字芯片无法正确工作，而你却没有查看一下是否为它提供了电源。它有时钟吗？假设你的图形硬件不工作了，应该检查以下问题：系统是否安装了正确的图形驱动程序？你是否运行了正确的操作系统？注册表中是否启用了这项功能？甚至还应该考虑是否运行了你要运行的代码？人们经常会说：“这段新代码运行起来与原来的代码一模一样。”随后却发现实际上根本就没有载入新代码。

你只是载入了旧代码，或者是载入了新代码，但系统仍然执行了旧代码，原因是你没有重启计算机，或者系统留下了一个很容易找到的旧代码的副本。

当我们看到一个问题时，通常在某个特定位置看到了问题，但导致这个问题的原因却在上游或者是一个基础性的问题。系统不具备正确操作的条件，于是出现了非常奇怪的行为。当你看到完全来自另一个世界的问题时，应该停下来，看看你是不是还在地球上。

在前面的VGA视频采集的案例故事中，我最后把问题归结为硬件的功能与文档记载不符。在证据面前，我没有假设硬件功能是正确的，相反，我联系了供应商，他们承认他们的功能出了问题。

假设你打开电视，屏幕上全是雪花点。你不会拆开电视修理它，而是首先怀疑是否接收到了良好的画面。你的VCR是不是选择并接收了3频道，而你把电视调到了7频道？或者，电视天线是否对准了佛蒙特州的East Snowshoe，而那里只有一个UHF电台？是不是有线电视公司又出了故障？或许你正要观看一场12月中旬举行的顶级的Bay Packers[①]比赛。但肯定不是电视的问题，而且你很走运，因为电视并没有用户可维修的零件，而且你当初在Best Buy买电视的时候，并没有理会售货员小伙子向你推销的3年全包维修合同。

你的苏芙蕾[②]没有膨胀起来，炉子打开了吗？

你的汽车无法发动。在你卸下化油器之前，先看看是不是没油了。

9.2 从头开始检查

在发动汽车时，另一个要考虑的方面是启动条件是否正确。可能电源插头插上了，但你是否按下了启动开关？图形驱动程序是否已初始化？芯片是否已复位？是否对注册表进行了正确的编程？在使用除草机之前，是否按了3次“primer”键？是否塞上了气门？是否已经

① Green Bay Packers是一支美国足球队的名字，总部设在美国威斯康星州的Green Bay。

② 苏芙蕾（Soufflé），法式甜点蛋糕。

把on/off开关设置到on的位置？（我经常在拉了六七次除草绳之后才注意到这一点。）

如果你的程序运行之前需要初始化内存，而你又没有显式地执行这个操作，那么情况会更糟。有时启动条件会是正确的，但当你向投资者演示的时候，它却出了错。

9.3　对工具进行测试



案例故事　在一个多媒体项目的早期，我们让一位顾问（他的佣金非常高）对处理器读写视频文件的速度进行基准测试。这是一个486处理器，主频是33-MHz，这在当时已经是非常快的速度了。我们需要达到一定的速度，而且想看看是否能保持这个速度。这位顾问的测试程序并没有我们希望的那么快，而且奇怪的是，他的读程序比写程序要慢，这与我们的经验不符，也出乎了我们的意料。他用了几星期时间来研究这个问题，尽其所能优化了代码循环的每个部分。最后，他承认他无法理解为什么读取的速度会更慢，他把结果提交给我们。

我们检查了他的代码，发现他并没显式地把文件数据类型设定为二进制（1和0），根据没有指定。在我们的软件开发环境中，这意味着数据类型自动被设置为文本输入（字母和数字）。这使得系统（在传输读取的数据时）在文件中搜索换行和回车，以便替代换行符，这极大地减慢了读取的速度。我们把函数修改为显式地传输二进制数据，读取功能的速度明显变快，这也符合了我们的预期。当我们询问这件事时，顾问回答说他认为在不指定文件类型的情况下开发系统默认是二进制的。但他并没有检查，他的假设是错误的。■

这位顾问在遇到无法解释的问题时，花费了几星期时间，并浪费了我们的大量资金，在本身并没有错误的代码中查找bug。但他从来没有考证过他的编译器是否按他想象的那样工作，因此永远也不会找到问题。而且他也永远失去了为我们工作的机会。

正如上面的案例故事所显示的，可能你对正在构建的产品所做的假设并没有错，而是对你所使用的工具做出了错误的假设。默认设置是一个常见的问题。另一个常见的问题是搞错了应用程序的环境。如果你的程序是使用Macintosh计算机开发的，显然无法在Intel PC上运行，但你的库和其他公用代码呢？一定要确保你的配置是正确的、最新的。开发工具不匹配可能导致一些非常奇怪的bug。

不仅仅是你对工具所做的假设可能有错误，而且工具本身也可能有bug。（实际上，你可能认为工具没有bug，而这种假设本身有可能就是错误的。工具是由工程师构建的，为什么它比你构建的软件更值得信任呢？）



案例故事 我们制作了一个专门定制的芯片，但它出现一个硬件错误，偶尔会丢失来自外设的中断信号。由于硬件工程师无法深入到芯片内部去查看发生了什么问题，因此他只能模拟问题。当然，芯片在模拟中工作得很好。但他的模拟是在寄存器逻辑层进行的，因此他决定看看芯片编译器在构建寄存器时到底执行了什么操作。他观察了更低层的门电路，发现编译器产生了一个时序问题，中断信号就是在这里丢失的。编译器报告它构建了可靠的寄存器——它的报告几乎就是正确的，唯一的区别就是有bug。■

你对调试工具所做的假设也有可能是错误的。当你使用一个没电的“连接测试仪”（continuity checker）来测试某个连接时，即使连接是好的，它也不会发出“嘟嘟”声，这会使你错误地认为这个连接有问题。因此首先要做的事情是连接两个探针，确定在不测试任何连接时它发出“嘟嘟”声。你需要对测试工具进行测试。在用示波器测量信号之前，先用一根手指接触探针，确定它有反应，然后再连接一个5伏的电压，确定它的垂直比例是正确的。当在软件中添加一段打印语句时，如果把它设置为仅当特定事件发生时才打印，那么当打印语句出问题时你永远也看不到这个事件。因此无论事件是否发生都要打印出一条消息，这条

消息只是说明事件是否发生即可。如果你给你的小孩测完体温的读数是75°F[①]，那么换个体温计再测一次。（如果还是75°F，那么让这个可怜的孩子从冷冻室里出来吧。）



案例故事　继续前面讲的那个老房子的故事，淋浴喷头的热水供应恢复正常几个月后，炉子突然不工作了。由于我们现在把柴炉换成了油炉，因此我想可能是没油了，我立即检查了油罐的压力表，显示还有1/4的油。于是我打电话请来了一位炉子维修工，他当天晚上就来了。他像我一样，立即去检查是不是缺油了。虽然我告诉他我已经检查过了，但他还是做了检查，并用手电筒敲了一下油表，油表慢慢回到了零。他帮我运来了一些油，而我感觉自己像是一个愚蠢的“城市书呆子”。■

> “深信不疑是真理的可怕敌人，甚至比谎言更为可怕。”
>
> ——弗里德里希·尼采

9.4　小结

检查插头

一些显而易见的假设往往是错误的。请恕我赘述，假设错误通常是最容易修复的错误。

- **置疑你的假设**。是否运行了正确的代码？是不是燃气用完了？插头是否已插好？
- **从头开始**。是否正确地对内存进行了初始化？是否按了除草机上的“primer bulb”按钮？开关是否已打开？
- **对工具进行测试**。是否运行了正确的编译器？燃料油表是否被粘住了？量表是不是没电了？

① 75华氏度约等于24摄氏度。

第 10 章

获得全新观点

10

“要想重新理清一个案子的头绪，最好的方法就是把它讲给别人听。”

——福尔摩斯，《银色马》



案例故事 我的车曾经出过一次毛病，这个问题非常奇怪，我甚至不记得准确的细节了，大概就是每当我倒车的时候，刹车灯的保险丝就会被烧断。在我意识到这是由倒车引起的之前，已经坏掉几个保险丝了，有一次我无意间跟一位很懂修车的同事聊起了这件事。他立刻说：“肯定是车内圆顶灯的一根线被挤压出来了，搭到了车身上。你把顶灯罩打开，把这根线拿开，用绝缘胶布缠一下就好了。”我大笑，顶灯跟刹车有什么关系呢？他说一直都有关系（参见图10-1）。

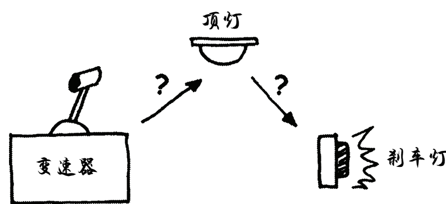


图10-1 汽车修理的大致方法

我断定这足够幽他一默的了，于是到车里拆下了顶灯罩。果然在灯座下面有一线被挤压出来了，接触了车身。我把它拉出来，用黑胶布缠上，然后倒车，看着刹车灯，它并没有熄灭。■

我也可以采用其他的规则来发现这个问题——阅读汽车的电路系统，并用电表一一测量各个地方的电线。但这将耗费大量时间，而我的同事有经验，他已经知道了答案。我需要做的就是请求他的帮助，这只需几分钟时间，而且，由于我正在上班，很方便就能拿到绝缘胶布。

10.1　寻求帮助

向别人寻求帮助至少有3个原因（还不算把整个问题甩给别人）：获得全新观点、专业知识和经验。而且，人们通常很愿意帮忙，因为这给了他们一个证明自己很聪明的机会。

10.1.1　获得全新观点

我们按照自己老一套的思路是很难看清全局的。我们都是普通人，对任何事情都有偏见，包括对bug隐藏在哪里的看法。这些偏见可能导致我们无法看清实际情况。而其他人则会从一个无偏见的角度来看问题（实际上他们只是有另一种不同的偏见），这可能会给我们很大的启发，帮助找到新的方法。即使无法从他们那里得到帮助，他们也可以安慰你一下，告诉你这个问题真是一个非常棘手的问题，也可以借给你肩膀靠一靠。

事实上，有时向别人解释问题也会使你有全新的认识，之后你自己就解决了问题。对事实进行组织的过程迫使你跳出你原来的思维模式。我甚至听说有一家公司在一个房间里摆放了一个人体模特，人们首先向它来解释自己的问题。我想这个人体模特一定非常有用，使很多问题得到了快速的解决。（也许它的互动性比你的同事还强。我打赌它也是一位非常宽容的听众，你的任何令人遗憾的误解都不会出现在下一年度的工资评审上。）

10.1.2　询问专家

有时候系统的某个部分看起来可能很神秘，这时我们不必到学校学上一年，而可以咨询专家来了解需要快速掌握哪些知识。但一定要找一位真正懂得你的问题的专家，如果他只是向你讲述一些晦涩难懂的时髦理论，那么他可能只是一个向你吹嘘技术的“江湖郎中”，而

不会提供帮助。如果他告诉你这需要花费30个小时，还要为你准备一份报告，那么他是一位顾问，也许可以为你提供帮助，但你需要付费。

在任何情况下，专家都比我们更“理解系统”，因此他们知道查找问题的大致路线图，也能够为我们的搜索工作提供很好的提示。当我们找到bug时，他们可以帮助我们设计一个正确的修复方案，以便不会影响系统的其他部分。

10.1.3 借鉴别人的经验

你可能经验不足，但你周围可能有人以前见过你遇到的情况，当你向他们快速描述事情的经过后，他们会准确地告诉你出了什么问题，就像上面案例故事中所讲的车内顶灯短路一样。专家难求，同样，具有某一特定领域经验的人可能也很难找到，因此需要高昂的费用，但这笔钱是值得的。



案例故事 这是一个古老的故事，讲的是有一位先生在一家大型工厂做设备维护工作，他工作了很多年，直到退休。他走之后，工厂有段时间运转得很好，直到有一天一台机器突然停止工作了，尽管新的维护人员尝试了各种各样的办法，它还是纹丝不动。他们打电话给这位退休的老员工，请他帮忙。他说他可以修好它，但需要10 000美元。工厂方面很着急，于是答应了他。

他带着工具箱来到工厂，走到机器旁，然后打开工具箱，拿出一把锤子，在机器的一侧敲了一下。机器开始运转了。他把锤子收起来，合上工具箱，然后索要他的10 000美元。工厂主很生气：“用锤子敲一下就值10 000美元吗？”“不，”他纠正他们，“敲一下只收10美元。知道在哪里敲击收9 990美元。”■

前言中曾提到，有些系统提供了故障维修指南，它收集了很多故障检修人员在修理某一系统时积累的经验。如果你正在使用的系统有这样一本指南，当你遇到“某个部件坏了”这样的问题，而非设计问题时，就可以求助于这本指南。只需在症状表中找到你的问题，就能

够快速地修复它。



案例故事　我设计的那款电视游戏开始生产后，技术人员开发出了他们自己的内部故障维修指南。球的运动是通过电容器（即用于盛装电荷的容器）的电压控制的。但公司购买了非常廉价的元件，很多电容器漏电，他们把这些电容装配到了主板上。技术人员很快发现当球向上运动的速度大于向下运动的速度时，需要更换电容A。当向左运动比向右运动快时，需要更换电容B。他们把这些记录下来，每当球速出了问题，立刻就更换相应的电容。■

10.2　到哪里寻求帮助

当你寻求帮助时，有很多资源可用，具体取决于你是想获得深入见解还是专业知识，或是经验，或者是其中的某几样。当然，你有一些同事，他们很聪明，可能是某个主题的专家，可能以前见过你遇到的问题。有些公司正在开发他们所说的知识管理系统，用于从文档和电子邮件收集信息（同时会注明这些信息是谁编写的），这样你就可以查询公司的知识，并了解谁掌握这些知识。本书写作时这还是一个全新的概念，但已经有一些公司这么做了，如果你的公司有一个这样的系统，要注意利用它。如果没有，就必须通过传统方式来查找信息了——搜索文档数据库并在咖啡机旁边请教你的同事。

如果你正在使用第三方供应商的设备或软件，那么可以给他们发送电子邮件或打电话。（还可以向供应商提交bug报告，他们会很欢迎。）通常，他们会告诉你一些常见的误解，记住，供应商既有产品的专业知识，又有经验。但有时供应商也没有经验，就像前面案例中讲的那个相位功能出错的VGA视频采集卡的例子，你发现了一个产品工程师也从未见过的新bug。但这些工程师所拥有的专业知识可以帮助确定bug是由什么问题产生的，从而使你摆脱问题的困扰。他们甚至可以给你一个修复方案，或者至少提供一种临时解决办法。在VGA的例子中，我在听到供应商的解释后，对我的代码做了调整，从而

解决了这个bug。

向供应商咨询并不总是意味着联系服务人员。有些公司没有客服人员，但大多数供应商至少会为你提供某种书面形式的帮助。这里再次强调“阅读手册”这条规则。事实上，这里的建议变成了“当所有其他方法都失败时，再次阅读手册”。是的，如果你严格遵守了第一条规则，那么已经读过手册了。现在带着你新发现的重点再次阅读一遍，或许你会看到并理解先前没有注意的内容。

一些积极提供服务的供应商通常会把信息发布到网上，你可以访问他们的网站，查找应用提示和样例程序。记住，还有其他用户会遇到与你一样的问题，找到他们并寻求帮助。查找一些资源站点，例如用户组留言板。Usenet新闻组的网址是http://www.dejanews.com/，你的因特网服务提供商也常常是一个丰富的用户信息交流资源。供应商的专家甚至会时时留意其中的一些站点，他们会回答一些重要的问题。如前所述，如果你正在解决一个常见系统的故障，应该查看正确的故障检修指南。

最后，还有很多资源提供了更基本和通用的知识，包括工具、编程语言和最佳设计实践方面的内容，甚至还有调试。（你在本书中学到的规则将帮助你更系统地运用从这些资源获得的信息。）你可以去当地的书店看看，或访问在线书店，订阅相关杂志和新闻邮件，也可以到网上进行搜索。专业知识无处不在。

10.3　放下面子

你可能害怕寻求帮助，你认为这是无能的表现。但事实恰恰相反，这只是表明你急于修复bug。如果你获取了正确的见解、专业知识和经验，将会更快地修复问题。这并不会暴露你的弱点，如果说有什么的话，也只是说明你明智地选择了帮助。

这个道理反过来也是成立的。不要认为自己很无能，而把专家看成是神。有时专家也会把事情弄错，如果你坚持认为自己是错误的，将会很糟糕。



案例故事　故事讲的是我编写的一个程序（有些读者喜欢刨根问底，那么我告诉你，它是一个用Forth语言编写的编辑器，用于处理用绕线[1]方法制作的布线板），它使用了一种叫做“B树”的大型数据库索引方案。我从一位同事已创建好的代码基础上开始工作，他是一个计算机科学迷，夜里经常枕着高德纳[2]的书睡觉。有一次，我在家里工作，查看他写的源代码（那时还没有个人电脑，因此我并没有实际运行代码），我突然遇到了一段无法理解的程序。看起来在某种特定的情况下，这段程序会删除一大块数据，同时重新调整树的平衡。我琢磨了几个小时，试着理解我是不是看错了，因为这个错误太明显了，我的同事不可能不注意到它，我想一定是我看漏了什么东西。

最后我还是在上班时把这段代码清单铺在他的桌子上，告诉他我看不懂这段代码是如何正确工作的，让他给我解释一下。

他看了几分钟，然后说“哦，这是一个bug。”■

10.4　报告症状，而不是理论

无论你想要获得什么样的帮助，在向别人描述问题的时候，一定要记住一件事：报告症状，而不要讲你的理论。之所以要从别人那里获得全新的观点，就是因为你的理论起不到任何作用。如果你找了一个人，把你的理论告诉他，那么也会把他拉到你原来的思维定式中。同时，你很有可能会把一些需要让他知道的关键细节隐藏起来了，因为你自己有偏见，认为这些细节不重要。因此一定要注意这一点。当寻求帮助时，描述发生的事情，描述你看到的一切。如果有可能，还要把条件描述清楚。告诉别人什么事情是间歇发生的，什么事情不是。但不要告诉他你认为问题的原因是什么。

---

① 绕线（wire wrap）是一种电子技术，它不使用印刷电路板来装配电子元件，流行于20世纪60年代和70年代初，后来主要用于短期使用和原型制作。

② 高德纳，世界顶级计算机科学家之一，著名的算法大师。

让他提出自己的观点。他们的观点可能与你的观点相符，也可能全然不同，而这正是你想要的。我曾经见过许多这样的错误，人们向别人寻求帮助，但那个人立即就被原来的无用的理论给“污染”了。（如果你的理论有什么用的话，就不需要找人帮忙了。）有些情况下，一个好的帮助者能够穿透所有这些迷雾，最后找到事实，但更常见的情况是，你只是把更多的人拉到你的思维定式中。

假设你后背的下方疼痛，你去看医生，医生想听你说说你感觉如何，而不想听你说你在因特网上查询了一番，断定自己得的是脊背的肿瘤。汽车修理工想知道你的汽车发动机在寒冷的早上无法启动，而不想听你说你认为这部通用公司的汽车是在星期一生产的，那位还没完全醒酒的装配工人不知把哪个传动装置拧得太紧了。

这条规则反过来也是适用的。如果你是帮助者，那么当向你寻求帮助的人讲起他的理论时，你一定要捂住耳朵，大喊“啦—啦—啦—啦—啦—啦……”，然后跑开，不要被他的理论所“污染”。

即使不是十分肯定，也可以提出来

有一些地方属于不好判断的“灰色地带”。有时你可能注意到一些数据看起来很别扭，像是错误的，或者与问题有某种关系，但你不确定为什么会这样。这些地方是值得提出来的，事实是你发现了一些出乎意料或不理解的事情。它可能与问题无关，但至少是有用的信息。记住，有时衬衫的图案和冰淇淋的口味也至关紧要。

10.5　小结

获得全新观点

不管怎样，你都需要休息一下，喝杯咖啡。

- **征求别人的意见**。甚至一个不说话的人体模特也能帮助你认识到你先前没有注意到的事情。

- **获取专业知识**。只有VGA视频采集卡的厂商才能够肯定相位功能发生了错误。
- **听取别人的经验**。别人会告诉你车内顶灯的线被挤压出来了。
- **帮助无处不在**。同事、供应商、网络，还有书店，都在等待着为你提供帮助。
- **放下面子**。bug发生了。以除掉bug为自豪，而不要非得以自己除掉bug才为自豪。
- **报告症状，而不要讲你的理论**。不要把别人拖进你的思维定式中。
- **你提出的问题不必十分肯定**。甚至连“穿了格子衬衫”这样的事情也可以提出来。

第 11 章

11

如果你不修复bug，它将依然存在

“当危险已经离你很近时，拒绝承认它并不是勇敢的表现，而是愚蠢。”

——福尔摩斯，《最后一案》



案例故事　在从加利福尼亚北部搬到东海岸之前，我买了辆旧车，开着它取道洛杉矶去东部。我走的是一条陡峭、漫长的山路，它通往洛杉矶北部的丘陵地带。为了保持爬坡的速度，我必须把档位挂到最低一档。但在我到达山顶之前，发动机突然熄火了。我踩住离合器，使车子滑行到应急车道，然后停下来。我一边抱怨一边想着该怎么办，最后没有任何好办法，只能试着再发动一次。我转动钥匙，车子正常发动了。我小心翼翼地驾着车走完了最后一段山路，想着它可能还会再次熄火，但它并没有。

在后面的旅途中，我驾车经过西弗吉尼亚的北部，当时已经是12月底，天气异常寒冷，天空中飘着干冷的雪花，我把车停在山腰处一个很小的加油站加油。随后又开始爬坡，车子再次熄火了，我把车滑行到路边，想起了在洛杉矶的那次抛锚，于是再次转动钥匙，车子仍然发动了。在剩下的旅程中车子没有再熄火。由于我并不是十分信任这家小加油站，我想可能是由于输油管里有水造成的（尽管这无法解释洛杉矶的那次事件）。我加了一些干燥剂，希望问题就此消失。

但问题并未消失。后来在那个冬天汽车再次抛锚了，这次是在一条完全平坦的路上，我正在高速行驶。我把车停到路边，立即试着重新打火，但车子并未发动。我等了一小会儿，再次尝试，这次汽车发动了。我平时也能够开着这辆车上班，但我发现每当加速到每小时45或50英里，不久车子就会熄火，然后我在路边停一两分钟，它会再次发动。

我并不是修车工，于是我把车开到当地的汽车修理部。他们说是电路的问题，换了一些电线，收了我75美元。当天车子照样再次熄火了。（这使我明白了你付给人们1小时50美元并不意味着他们知道如何调试问题。这也是在我的工程生涯早期得到的一个教训。）

我前思后想，这几次事件都发生在我把档位调到最低档的时候，要么是因为爬坡，要么是因为我要加速。当车子抛锚之后，只要等一小会儿，它就会再次发动。我非常熟悉发动机，我知道发动机由化油器给油，化油器有一个很小的储油器，而它的油是由油箱供应的，油箱是一个大的储油器。我想如果有某个东西阻碍了油从油箱流到化油器，那么当我调低档位时，化油器中的油很快就会用光，然后我就必须等待油慢慢从油箱流进化油器，重新把它注满，然后发动机才可以重新发动（参见图11-1）。

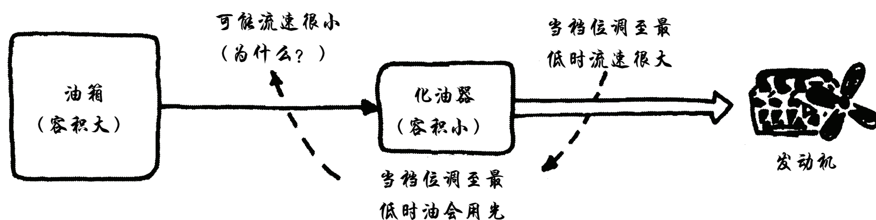


图11-1　化油器是一个真正的“汽油守财奴”

于是，我采用了“获得全新观点”这条规则，在上班的时候到咖啡机旁边问了很多人这样一个问题：“什么东西会限制油从油箱流到化油器？”一位知识渊博的朋友告诉我“是滤油嘴太脏了”。于是我花50美分买了一个滤油嘴，自己把它换上，问题解决了。■

汽车修理部并不知道他们是否已经解决了我的问题。修车工并没有制造失败，在修理之后也没有验证问题是否不再发生。（但他们收下了我的75美元。然而，这却是他们从我身上赚到的最后一笔钱了。）

11.1　检查问题确实已被修复

如果你遵守了“制造失败”这条规则，就会知道如何验证你确实已经修复了问题。那么应该立即验证！不要假设问题已被修复，而要测试它。无论问题和修复看起来多么明显，你都无法保证修复是有效的，直到做了测试。你可以向一位年轻人收费75美元，但如果问题仍然存在，他会不高兴。

11.2　检查确实是修复措施解决了问题

当你认为你已经修复了一个设计问题时，取消这个修复，确定系统再次失败。然后再应用这个修复，再次验证问题已修复。直到你经过从修复到失败，再从失败到修复这个过程之后（只应用和取消修复，而不改变其他地方），才能够证明你确实已经修复了问题。

你可能会问：“为什么一定要这么做呢？”因为在调试期间，你往往会改变一些不属于“修复”的东西。可能是一个测试序列，也有可能是软件或硬件的某一部分。或者只是有一些随机因素不同了。如果这些更改对问题没有影响，那当然没关系，但有时它会修复或隐藏问题。你并没有意识到这一点，而只是对你做的修复做了测试，发现它起作用了，于是你高高兴兴地回家了，但你所做的修复与问题的消失毫无关系。如果你把这个修复方案发给客户，他们并没有像你一样改变别的地方，那么系统将再次失败，这是非常糟糕的。

如果只把修复撤销，系统将仍像过去那样发生失败，那么你就可以非常肯定测试序列并没有被改变，你的修复确实解决了问题。



案例故事　《芝麻街》中有这样一段剧情：SuperGrover和Betty Lou都在尝试打开一台计算机。Grover说："嗯，或许我一边蹦跳着一边大喊'Wubba!'它就会打开了。"于是他就一边跳着一边喊："Wubba! Wubba! Wubba!"正巧这时Betty Lou找到了ON按钮，她按了这个按钮，计算机打开了。Grover完全没有注意到Betty Lou的行动，他看到计算机启动了，于是断定自己发现了一种非常有价值的计算机修理技术。（此处的引用已得到芝麻街工作室的许可。）■

当然，在有些情况下，如果你只是修理一个设备（而不是工程设计问题），那么重新制造失败可能是不必要的，也是不便的。我没有必要把脏的滤油嘴再放回车里。把一个心脏移植病人原来的心脏再装回去不但没有必要，而且很危险。不要干蠢事。

11.3　bug 从来不会自己消失

如果你不修复它，它不会自动修复。每个人都希望看到bug消失。"看起来它不会再出问题了。""这个问题发生了几次，但后来不知道发生了什么，它不再失败了。"当然，逻辑上的推断是"或许它不会再发生了。"但事实上它仍会发生。

假设你先前曾经使系统发生过故障，由于以某种方式改变了条件，它不再出故障（或者故障频率变低了）。如果你完全靠凭空猜测，并修改很多地方（就像用散弹枪射击一样），那么或许你碰巧真的会解决问题，但你当然不知道问题是如何解决的，可能你也不想用你下个月的薪水来打赌问题已修复。当然，你更不想贸然地赌一把公司的下一个产品不会发生这个问题。

因此，回到第4章，重新读一下如何使间歇性故障更有规律地发生。返回最初的系统和引入bug的测试场景。如果用旧软件可以使bug发生，而当使用最新版本的软件时问题不见了，那么你可能已经修复了问题，这时应该查看一下新旧软件的区别，并找出失败

的原因。

有时你没有足够的时间。如果你不得不说“伙计，现在要想使这个问题再发生一次实在太难了，我们没法修复它了”，那么你晚上也不会睡个踏实觉。可以肯定的是，一些客户会遇到问题，这证明你的失眠不无道理。因此，要有所作为。在系统中植入某入插装工具，如果产品在客户现场真的发生故障，它可以捕获一些信息。如果问题永远也不发生，插装工具也不会有什么妨碍，而当发生问题时，你就可以“制造失败”，而且有一些信息可供查看。

这样做还有一个额外的好处，那就是即使你无法利用这些信息来修复问题，至少客户会知道你认真地跟踪了问题。当他们报告问题时，你可以告诉他们：“非常感谢！几个月来我们一直在尝试捕获这个极为罕见的问题，请把日志文件发送到我们的邮箱。”这样说要比“哇，真是难以置信。我们这里从来没有发生过这个问题”要好得多了。

11.4　从根本上解决问题

如果一个硬件设备失败了，不要以为它是无缘无故坏掉了。如果在某种条件下有零件会损坏，那么更换这个零件也只能是为你换来很短的时间（如果有的话），然后新的零件也会损坏。你必须找到真正的失败之处。在扬声器短路的那个案例故事中，是线路的问题导致右声道的保险丝被烧断。通过更换保险丝（把左声道的扬声器换过来）只会导致这根保险丝也被烧断。遗憾的是，他们只有这一根保险丝。



案例故事　我最不愉快的一次调试经历发生在我上大学的时候。我家人的一位朋友给了我一台集成的立体声系统，它带有一个扩音器、一个AM/FM收音机和一个8音轨的磁带卡座（是的，那是很久以前的事了）。但有一个问题：它已经不再工作，打开电源时不会发出任何声音。

我把它拿到学校，找出了万用表，开始检查电源系统。我测量了从变压器引出的四五条线，都没有电压，而插座是有电的。于是我断定是变压器坏了。当然，这台音响没有任何说明书，也没有好用的变压器，我不知道它的输出电压应该是多大，因此不能随便更换一个别的变压器。于是我把变压器的型号抄下来，从厂家订购了一台新的变压器。

几个月过去了，变压器终于到了。我打开立体声音响，高兴地发现线的颜色完全匹配，于是把旧的变压器换下来。我打开音响，调到WBCN（当时是波士顿摇滚音乐电台），开始享受音乐。我并没有测量线路的电压。

大约一个小时后，我们当中有几个人决定到WBCN电台去拜访一下（它就在Prudential大厦的顶层，与我们的宿舍只隔几个街区）。我们打算在播音室的大玻璃窗外举一块有趣的牌子，请求主持人为我们播放一首歌曲。当我们回来时，留下的两个人告诉我们DJ可能说了我们的请求了，也可能播放了我们点播的歌曲。但他们并没有听到，因为在我们走后不久，音响就开始冒烟，烧坏了。

后来，我拆下了被烧坏的变压器，很懊悔当初没有测量电压。如果我知道了电压，下一次就可以用更容易买到的变压器来更换。这次，我测量了连接到变压器上的电路，发现8音轨的磁带卡座短路了。虽然新的变压器未能幸免，但它工作了足够长的时间，使我误认为我已经解决了问题。

我没有再买新的变压器，而是直接扔掉了这台音响。■

显然，在这次经历中，我违反了很多原则。我没有理解系统，也没有进行足够多的观察。当我打开电源而发现它没有任何反应时，我认为已经完成了“制造失败”的过程。实际上，当我用新的变压器来驱动短路的电路时，我才真正制造了失败，但在那个时候没有测量任何东西。最重要的是，那是因为我根本没考虑到变压器的损坏可能是由于其他条件造成的。

11.5 对过程进行修复

前面讲过本书不打算涉及质量过程，但有时候修复系统和修复过程（正是这个过程导致了bug）之间的界限很难分清楚。跨越这条界限也许是好事。这里有一个例子：在一家工厂里，油漏到了地上。你的解决办法可以是把油擦干净，但这并没有解决问题的根源——如果有一个设备漏油，它仍会继续泄漏。因此，你可以把它固定得更紧一些。但问题修复了吗？没有，它会再次变松，因为机器振动得很厉害。这是由于它只是用两个螺栓固定的，而没有用4个。看，这才是真正的漏油bug。我们遵循了上一节的建议，追踪了导致问题的基本条件。

事情并未到此结束。由于下一台、再下一台机器仍然只使用两个螺栓来固定，因此还会有油漏到地上。你必须修复设计过程，确保在需求、设计和测试阶段正确考虑振动。

虽然这是一个设计质量的问题，但我一直把ISO–9000看做是一种对设计过程保持审计跟踪的方法。在这个例子中，我们要找的bug隐藏在设计过程中（忽略了振动），而不是在产品中（漏油的设备），但审计跟踪的工作原理是一样的。正如前言中所讲的那样，本书介绍的方法完全是通用的。

11.6 小结

如果你不修复bug，它将依然存在

现在你已经掌握了所有的技术，没有理由再让bug存在了。

- **查证问题确实已被修复**。不要假设是电路的问题，而仍然让汽车带着脏的滤油嘴上路。
- **查证确实是你的修复措施解决了问题**。口中大喊“Wubba!”并不是使计算机打开的窍门。

- **要知道，bug从来不会自己消失**。使用最初导致它失败的方法再次制造失败。如果必须交付产品，那么就在产品中设计一个用于捕捉bug的“陷阱”，以便产品在客户现场发生失败时，把它捉住。
- **从根本上解决问题**。在烧坏另一台变压器之前，先把无用的8音轨磁带卡座扔掉。
- **对过程进行修复**。不要只是擦掉地上的油，而要纠正设计机器的方式。

第 12 章

通过一个案例讲述所有规则

“你了解我的方法。它建立在对琐事的细微观察之上。”

——福尔摩斯，《博斯科姆比溪谷秘案》



案例故事 公司T有一台小设备，它有时会拒绝启动，因为有个微处理器无法正确读取使用备用电池供电的（battery-backed-up）内存。有些内存单元的情况比其他单元更糟，如果一个单元某次启动时失败，下次就无法再工作。这意味着内存中的数据没问题——问题在于读取它。

工程师A对此问题进行了研究，得出的结论是问题是由于某种数据总线噪声引起的，导致无法正确读取内存。系统是一个包含内存和一个使用备用电池供电的内存控制器的小型电路板，它被添加到一块现有的主板上。该电路板与主板之间的连接器只有两个接地引脚和一个5伏引脚，因此为了降低噪声，工程师在两块板之间增加了一根很粗的接地线。他还增加了一个电容器，作为小板上的蓄电池。他还写了一张工程变更清单，并获得批准，然后将改动应用于制造过程。

当第一批经过改动的电路板生产出来时，工程师B接到了电话通知，因为在这批主板中，很多使用效果与以前完全一样。

工程师B早上9点开始工作，他把示波器接到数据线上，然后观察当主板试图访问内存时会出现什么情况。当系统出现故障时，噪声并未给数据造成多大损坏，

但数据都是1。查看读取脉冲的情况时，他惊讶地发现根本没有读取脉冲。现在，这是一个严重的信号丢失问题，而不仅仅是噪声。主板上的微处理器正在进行读取，并发送一个读取脉冲给内存控制器芯片，但是读取脉冲并未从内存控制器芯片中出来并到达内存（参见图12-1）。

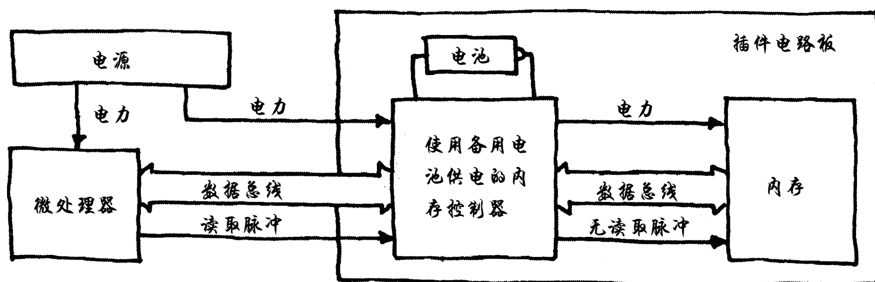


图12-1　读取脉冲丢失的案例

工程师B快速查阅了手册，发现内存控制器芯片的作用是，当电源电力不足时防止访问内存（因此也屏蔽了读取脉冲）。这看起来很有道理，但当时电源似乎没有出现明显的问题。

上午9:45，工程师打电话给芯片制造商，与一名应用工程师进行了沟通。该工程师说："哦，你可能在5伏电源和芯片电源之间安放了一个二极管。如果你这么做，当供给5伏的电压时，芯片就会认为电力不足，从而锁定。"毫无疑问，他们的设计与这名应用工程师描述的完全一样。（因为后面给电路板增加了元件，同时未对主板进行任何改动，这样就修改了芯片制造商推荐的设计，在当时看来这是一种非常合理的方式。）

按照应用工程师建议，工程师B需要将主板的另一条线连上，才能获得原始的5伏电压。当他这样做的时候，系统就工作正常了。他将修复还原，目睹它发生故障，然后再次进行同样的修复，接着进行测试。一切正常。10:15，工程师B圆满完成了所有工作。（在这次修复过程中，编写工程变更清单所花的时间确实比调试过程要长。）■

让我们总结一下这个案例中的规则。

理解系统。工程师A从头至尾都没有看数据手册。工程师B看过了数据手册，而且当他在其中没找到读取脉冲消失的原因时，他知道芯片有可能是“嫌疑犯”，因此心里很清楚应该联系哪家厂商。他也马上知道，没有读取脉冲会导致数据全部为1。

制造失败。系统在某种程度上有规律地出现故障的事实，会使工程师B的工作变得轻松。（同时会让工程师A的处境变得尴尬。）工程师B看到了数据都为1和读取脉冲丢失。

不要想，而要看。工程师A从未看到数据全部为1，也没有看到读取脉冲丢失，因此不可能很快知道这不是一个噪声问题。

分而治之。工程师B查看了接口，发现了错误数据。他接着查看内存读取脉冲，发现它丢失了，因此他顺藤摸瓜，发现了微处理器脉冲没有正确地到达电路板。最后找出了正常读取脉冲与丢失的读取脉冲之间的出错的芯片。

一次只修改一个地方。尽管工程师B怀疑另一位工程师的改动没有起到作用，但在测试时还是保留了这些改动——系统是因为已安装的改动而引发故障的，因此这些改动正是测试的目标。

保持审计跟踪。工程师B没有找到任何说明工程师A认为问题出在噪声上的信息，也没有找到工程师A对他自己的修复所做的测试结果。或许工程师A保存了审计跟踪记录，但他留作自用了。制造过程确实需要保持审计跟踪。故障报告充分证明了内存中的数据没有错误，因为它有时无需重新加载内存也能工作。这使得工程师B能够集中精力阅读函数，从而很快找出错误的数据和丢失的读取脉冲。制造过程的测试结果也清楚地表明噪声修复并不能解决问题。工程师B记下了所有内容，包括芯片厂商中那位提供了帮助的应用工程师的姓名。

检查插头。芯片的行为很有意思。工程师B觉得没有理由，因为他见过很多出现故障的芯片，而这块芯片很可能没有坏。他怀疑芯片的使用是否正确，而且非常肯定这是一个微妙的电源问题。

获得全新观点。但他不知道芯片的使用是否有错误。因此他咨询了一位专家。专家知道答案，而且立即告诉了工程师B。

如果你不修复bug，它将依然存在。工程师A显然没有很好地测试他的修复，因为他的修复没起作用。这种尴尬的失败给了工程师B一个很好的理由，让他在编写他的工程变更清单之前，一定要确保他的修复是成功的。

第13章

牛刀小试

13

“有人发明，就有人能看懂。”

——福尔摩斯，《跳舞的人》

你能说出以下这些调试场景中运用或违反了哪些调试规则吗？每个故事结尾针对相应标号处的场景一一给出了答案。

13.1 灯和吸尘器的故事



案例故事 这是我前面讲过的房子布线的调试故事。这个故事真的很奇怪，它使我回想起我老爸的名言：“当所有手段都不起作用时，去读说明书。”

在前文提过的位于新罕布什尔州已经有90年历史的老房子里，我们正在为我父亲的造访做准备，我妻子将吸尘器插头插入餐厅墙面的一个插座中。她后来告诉我，那个插座有点问题，因为她打开吸尘器时看到了一道闪光(1)。当然，出于好奇，我也把吸尘器接上，踩了一下踏板，同样看到了闪光(2)，我很快将吸尘器关掉。我意识到，这道闪光不像是电火花，而更像是房间的灯光。

我好奇地再次踩了一下踏板(3)，我头上的吊灯亮了。但吸尘器却没反应。我发现了这个很有趣的现象，并决定等我父亲来弄清楚原因(4)。当然，他也不太相信会

有这种事情。(他的原话是“这是不可能的!”[5]) 我们给他演示了一遍[6]，表明我们可以通过吸尘器的脚踏板开关来控制吊灯。

我们思考了一会吊灯的工作原理[7]。吊灯由两个开关控制，分别位于房间的两端。电从主电线进入其中一个开关，并通过两条电线之一连接到另一个开关。另一个开关将这两条电线之一连接到它那侧的一个普通的闭合开关，然后经过灯，再回到地线。如果这两个开关指向同一条电线，电路就是完整的，灯就会亮。如果你将开关转向，让它们指向不同的电线，电路就是断开的。按下任一开关都可以打开或关闭灯。

我们猜测[8]，插座与两个开关之间电路的接线不对。毫无疑问，当我们沿着电线下到地下室时[9]，我们发现了两个紧挨着的电气接线盒，一个位于主电源上，一个位于开关电路上。吸尘器使用的插座已经被连接到了错误的接线盒上，当然，最后和两条开关线搭在了一起（参见图13-1）。当我们打开吸尘器时，它使开关之间的电路闭合，从而打开灯。但是灯会用光所有的电压，因此没有足够的电压去打开吸尘器的电动机。

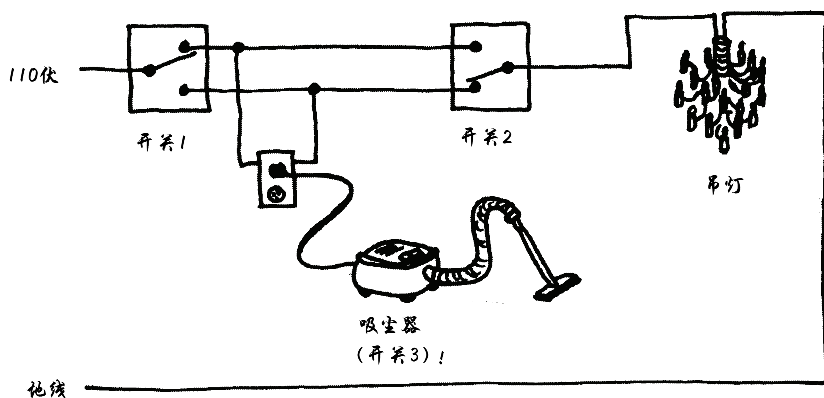


图13-1　一个三向的开关

在老房子里面布线是一件非常有意思的事情，当然，前提是不会把房子烧掉。■

已应用和被忽略（用括号括起来的部分）的规则。

(1) **保持审计跟踪**。我的妻子没有写下来，而她所看到的正是调试过程中最重要的事情。但是她记住了看到的现象，而且不是仅仅说“电源插座坏了”。她还注意到只有那个插座是坏的。

(2) **制造失败**。我亲自看了看，而不是急着重新接线。

(3) **制造失败**。我发现我第一次的测试结果难以置信，因此再次进行了测试。两次尝试均告失败，从统计学上讲这看起来相当可靠。

(4) **获得全新观点**。我让我父亲参与进来，很大程度上不是因为我被难倒了，而是因为我想看看当我向他描述问题时他脸上的表情。这通常是向另一位工程师咨询一个古怪bug的最好理由——应该与同样具有好奇心的人分享有趣的事情。

(5)（**不要想，而要看；制造失败**。）“这不可能发生”只是某些人的口头禅，他们想当然地认为某事不可能发生，而并未实地考查。

(6) **制造失败**。我们很快通过行动让我父亲确信这件事情确实发生了，而且屡试不爽。

(7) **理解系统**。这是一个房子布线的问题，因此我们分析了吊灯电路的工作原理。了解开关十分关键，这使我们决定应该查看什么地方。

(8) **不要想，而要看**。我们进行了猜测，但目的只是为了重点研究开关之间的布线。

(9) **不要想，而要看**。我们追踪了电线，然后亲眼目睹了问题所在。

13.2　大量出现的 bug



案例故事　1977年，我们正在做一个项目，它使用了1/4兆的内存（这么大的内存在当时可是前所未有的），但遗憾的是它不能一直正确读取。我们的一名软件人员编写了一个测试程序[(1)]，它把字符F（我们公司的首字母）的计算机代码加载到内存中，然后从连续的内存位置中读取字符，再把它们发送到一个视频终端上

显示出来[2]。

当时还是视频终端而非PC的天下。有些年轻的读者可能不知道，视频终端看起来像是PC机，但它没有主机，只能显示一行一行的文本字符，显示的时候字符从左至右排列在屏幕的底部。当需要在底部显示一个新行时，它会将整个屏幕向上滚动。

这是一个速度相当快的程序，因此字符在终端上显示得很快。很快整个屏幕上就填满了从底部向上滚动的F。随着F行数的上升，以前的行就看不到了，而每一行与它上面的行是完全相同的。看起来就像是满屏F保持不动，实际上所有F行都在快速向上移动。

出现错误时，就会出现一个不同的字符V[3]。V将出现在底部，然后快速上升到顶部。它就像是一只小鸟受惊后直接向上飞走（参见图13-2）。

图13-2　受惊的V

> 我们怀疑内存中存在噪声问题[(4)]，而能够让断断续续的噪声更加强烈的一种办法就是伸出一根手指触摸出现噪声的电路。我知道在8块内存芯片中应该查看哪一个[(5)]，是因为F的字符代码是01000110，而V的字符代码是01010110；显然，错误出现在第4位，当V出现时该位上的值为1。但我触摸内存芯片上的一组引脚时，屏幕上看起来就像是一大群小鸟四处乱飞，同时发出枪击一样的声音[(6)]，大量V字从屏幕底部向上滚动（参见图13-2）。
>
> 我将手指移开，它立即就安静下来[(7)]。接下来，我改进了我的研究[(8)]，因为我的手指并不十分精确。我找了一小段电线，拿在手中，然后逐个接触了上述的8个引脚。我找到了引起群鸟乱飞的引脚[(9)]，并使用示波器查看了它的信号[(10)]。这个引脚接的是内存控制线，而且我听到[(11)]它发出讨厌的响声。
>
> 结果证明，这条线过长（电路板的规格是18" × 24"）了，因此尾端需要一个终止器（电阻器，而非施瓦辛格）来清理信号。系统经过测试，使用终止器后不再出现群鸟乱飞的现象，把终止器拿掉便再次出现，接着再次把它放回，从而验证修复[(12)]。我们还在所有芯片上的其他信号线尾端都加了终止器[(13)]，因为这种问题在一块芯片上间歇出现，以后也可能出现在其他芯片上。■

已应用和被忽略的规则。

(1) **制造失败**。测试程序通过一个速度很快的循环对系统进行试验，因此错误大概每20秒出现一次，而不是1小时一次。

(2) **制造失败**；**不要想，而要看**。测试不仅加快了错误出现的频率，而且几乎在错误出现的同时显示了一个标记。

(3) **制造失败**；**不要想，而要看**。除了能够看到错误发生之外，我们还看到了错误的字符是什么。这个细节在后面变得很重要。

(4) **理解系统**。间歇出现错误的新硬件设计经常受到噪声问题的困扰（一条电线上的寄生电压将1变为0，或者将0变为1）。

(5) **理解系统**。我们知道字符的计算机代码是什么，而且我们知道哪块内存芯片用于保存第4位。（顺便对年轻读者说一下，过去很大的内存芯片的容量却只有1位。）

(6) **制造失败**。通过将更多噪声引入系统中，我能够更加频繁地制造失败。但这有点靠运气的成分。对于某些电路，用一个手指去触摸就像是使用一根消防水带测试漏窗一样——噪声将会使完全没有问题的电路出现故障。有时会事与愿违，手指的电容会消除噪声，让系统安静下来。（有这样一个传说，20世纪70年代，在麻省理工学院的模拟电路设计实验室，助教们设计出了一个在电子效果上等同于手指的电路——遇到噪声问题时，他们会接触电路中的各个节点，直到噪声消失为止，然后把手指电路焊接到这个节点上来解决问题。）

(7) **制造失败**。我肯定我的手指是错误爆发的原因，这一点在我将手指从电路上移开之后就能看出来。

(8) **分而治之**。我已经将问题的范围缩小到4到5个引脚上，现在我必须找出真正的罪魁祸首。

(9) **一次只改一个地方**。我证实了错误不是两个引脚通过我的手指互相作用的结果。

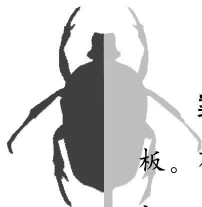
(10) **不要想，而要看**。我观察了信号，想查明发生了什么事情。我先假设是噪声的问题，但我并不急于下结论，而是先看看。

(11) **不要想，而要看**。我没有听到噪声，而是听到它发出响声，这两种属于不同问题，修复的方法也不相同。响声的确让系统更加容易产生噪声，这也是目前的常规噪声会使它间歇性地出现故障，以及来自我手指的噪声使它甚至更加频繁地出现故障的原因。如果我假定是噪声问题，并以此为依据进行修复，将会使问题变得更加不规律，但问题依然一直存在。

(12) **如果你不修复bug，它将依然存在**。我们确定问题已经解决，无论我的手指是否触摸电路。我们还确定当我们移除修复时，问题将卷土重来，同样无论我的手指伸到哪里。

(13) **如果你不修复bug，它将依然存在**。我们知道错误的根源在于糟糕的设计，而且其他信号和芯片也是这样设计的。我们修复了错误根源，以防止未来出现问题。

13.3 宽松的限制



案例故事 我们公司生产一种用于把PC机组装为视频会议系统的电路板。有些系统由两块电路板组成：一块用于处理音频/视频压缩和视频会议协议，另一块用于连接通信网络。我们有两种通信电路板：一种直接连接到电话公司的ISDN（综合业务数字网），另一种用于连接一种称为V.35的串行接口，该接口先挂接到一个客户服务单元（CSU），然后再接入电话公司（参见图13-3）。我们同时销售两种组合给客户，反响都很不错。

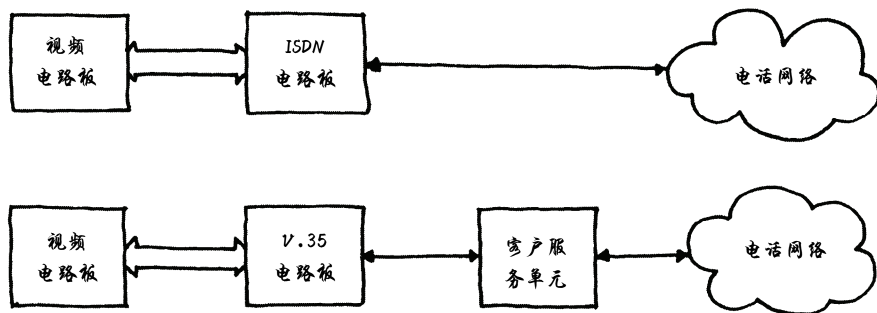


图13-3 ISDN和V.35

我们设计了一个新版本的视频会议电路板，其使用的通信接口与老版本相同，因此应该只要将它插到通信电路板中就能工作。但是质检人员发现，新的视频电路板无法通过V.35电路板进行某种类型的呼叫。它每次都会失败[(1)]——没有一次呼叫成功。ISDN电路板工作情况良好。

失败的呼叫属于受限呼叫，即允许电话公司使用老式遗留交换设备的一种特殊类型的呼叫。过去，电话公司在8位信道中通过网络发出呼叫，但使用其中一位实现内部用途。端点设备可以使用其他的7位。现代的电话公司交换机提供“纯信道”（clear channel）呼叫，因此端点设备可以使用全部的8位。在

视频会议中，我们尽可能地使用了纯信道，但有时无法获得纯信道的路径，因为电话公司只能通过遗留设备传送我们的呼叫。出现这种情况时，视频会议系统就会进行受限呼叫，并将所有内容挤到7位中。

视频电路板与通信电路板之间的接口发送的数据始终是8位，但如果呼叫受到限制，视频电路板可能只能使用7位。V.35电路板将会剥离未使用的位，只将正常的7位数据发送给CSU（参见图13-4）。

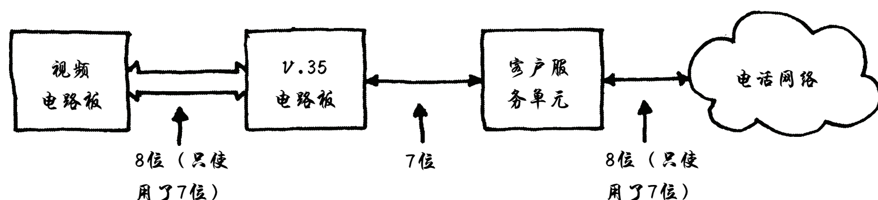


图13-4 受限的V.35呼叫

因为ISDN电路板工作正常，而且软件不关心挂接的是哪种电路板，因此软件工程师得出结论说，问题一定是出在视频电路板硬件中(2)，因为它不得不处理ISDN与V.35电路板之间的细微差别。硬件工程师坚持说，因为V.35所呼叫的纯信道工作正常，而且无论是使用受限信道还是纯信道，视频电路板硬件的工作方式都完全相同，因此问题一定是出在软件中(3)。

调试日志告诉我们，视频电路板无法在进入的位中找到帧指示位（完成呼叫时需要它）(4)。一位软件人员通过在进入的数据缓冲器上增加插装工具(5)，证明了没有来自硬件的帧指示位进入。接下来，我被软件组请去查找硬件问题，因为我了解视频帧指示位协议和V.35硬件(6)。

我首先使用软件工程师的数据缓冲器插装，确认了缺少进入的帧指示位(7)这个事实。然后，我在两张卡之间的硬件接口上安装了一台示波器(8)，我观察了输出的位(9)，意外地在第8位上发现了帧指示位。这一位应该是未使用的，并被V.35卡所剥离。帧指示位应该位于第7位中。

我们找到了为输出帧指示位选择位的函数，并在每次调用它时增加了一个

调试日志输出[10]。我们看到接口被错误地设置为第8位，找到设置它的错误代码，并修复了这段代码；V.35卡工作正常。当然，我们取消了修复，看到它又出现故障，然后再把修复原样恢复，以此来确认修复的正确性[11]。

新的视频卡软件在设计的时候就已经包含这个bug了，这也是新卡无法像旧卡那样对V.35电路板正常工作的原因。我们想知道为什么ISDN卡能够正常工作，因为软件确实不管使用什么卡都会设置错误的位。我们通过观察[12]与已插入的ISDN卡之间的相同卡间接口确认了这一点——协议最初是在第8位中发送帧指示位的。但很快它就转移到了第7位。

结果证明，当协议已经建立帧指示位并能够与另一端进行通信之后，它会发送这样一个命令："这是一次受限呼叫。"另一端看到这个命令之后，作为响应，就会把帧指示位移动到第7位上去。在使用ISDN电路板的情况下，两端都获得帧指示位，发送受限的命令，并在建立初始呼叫后进入受限模式。

"等一等！"你大声说，"这是一次受限呼叫，ISDN卡如何首先建立帧指示位呢？"我们沉着地回答："是什么使你认为呼叫真是受限的呢？"[13]我们的质检部门使用一台内部交换机为测试提供ISDN服务，因此当自动测试昨天整晚运行的数千次呼叫时，我们并没有给电话公司支付任何费用。显然，这台交换机完美地模仿了受限呼叫，正确地完成了所有的信号处理，除了它认为没有必要真正去掉这个第8位[14]。ISDN卡也将所有的8位都传给了这台仿真设备（参见图13-5）。因此仿真设备让ISDN系统将帧指示位问题留待它让自己进入受限模式中之后再解决。V.35电路板只发送了7位，不能像这样"作弊"。

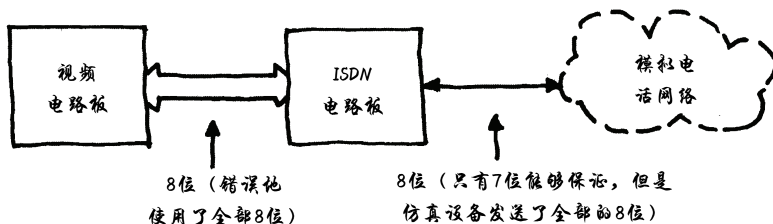


图13-5　宽松限制的ISDN呼叫

> 我不知道先前是否对受限模式的真实ISDN线路进行过任何测试。但要么是没有进行测试，要么就是电话公司在能够做到的情况下传输了第8位（尽管这没有必要）。无论是哪一种情况，bug都是存在的，亟待有人去发现。如果在实际的受限的电话公司电路中使用了ISDN系统，它就会像V.35系统一样出现故障(15)。■

已应用和被忽略的规则。

(1) **制造失败**。我们遇上了百分之百的失败，这是件十分幸运的事情。因为这条规则非常适用，问题应该很容易解决。缺少其他规则将会增加它的难度。

(2)（**不要想，而要看**。）这是一种猜测，这很不好，因为这会分散所有人查看问题的注意力。这也是一种典型的指责。

(3)（**不要想，而要看**。）以牙还牙的一种指责。虽然这不无道理，但是如果不亲眼看到是没有发言权的。他们不看，因为问题“明显”不在他们的专业知识范围内。这是想当然地相互指责的最坏情况，这实际上阻挠了人们的亲自观察。

(4) **不要想，而要看**。这一步查看能够告诉我们系统出现故障的确切时间（在呼叫开始时），以及呼叫过程的哪个环节有问题。它有助于引导我们进行下一步的查看。

(5) **不要想，而要看**（**分而治之**；**不要想，而要看**）。一位工程师看了一下，但只是证明了硬件没有提供帧指示位数据给软件。他没有追溯到数据的源头，那里才是问题的根源所在。他认为没有这个必要，因为他认为硬件才是问题所在。

(6) **理解系统**。我不仅熟悉硬件和软件，而且我没有任何理由指责其他任何人。（出现问题时的相互指责一直让我吃惊。我宁愿bug出现在我的工作范围内，这样我就可以修复它。我们同坐在一条船上，若别人所坐的一端漏水，我们也难逃沉船的厄运。）

(7) **不要想，而要看**。我想亲自看一看插装，因为我比头一个查看它的工程师更加了解协议。

(8) **分而治之**。他查看了从信道接收数据的软件。我则向硬件接口的上游移动。

(9) **分而治之。**进入的硬件流是错误的，因此我决定向上游追查。因为这是一个双向的通信系统，我可以向上游依次检查V.35电路板、CSU、电话公司、另一端的CSU、另一端的V.35电路板，最终到达另一个系统的两块电路板之间的接口，但现在是在出站而非入站的信道上。我承认，我猜想问题是对称的，于是查看了离我较近的系统的出站频道（所有的接头电线都可以看到）。如果它没有问题，我就可以转而检查到另一个系统。

(10) **分而治之；不要想，而要看。**我们向上追溯到生成协议的软件，利用插装来查看可能影响协议的因素，并发现受限模式的设置不正确。

(11) **如果你不修复bug，它将依然存在。**我们确保修复问题的过程能够真正修复bug。

(12) **如果你不修复bug，它将依然存在；不要想，而要看。**ISDN电路板为什么没有出现故障这个吹毛求疵的问题，给修复带来了疑问。我们想要理解正在发生的事情，以便确保我们真正修复了问题。因此我们查看了ISDN系统。

(13)（**检查插头。**）这个错误的假设是原来错误诊断问题的根本原因。我们假定测试是有效的，而且它证明了ISDN卡工作正常。

(14)（**检查插头。**）负责交换机的人员认为没有必要真正查看第8位。他们假定，发送这一位不会对任何人的测试造成影响——毕竟，要测试的单元并未使用这一位，对吗？

(15)（**如果你不修复bug，它将依然存在。**）现场可能会出现受限的ISDN呼叫，并将导致出现故障。如果它在现场坏掉，我们将不会知道。因此我们是很幸运的。

13.4 识破 bug



案例故事 我们正在开发一种顶部带有模拟触摸屏的手持显示器。这个触摸屏向计算机提供两个电压值，分别用于表示被触摸点的X和Y坐标（参见图13-6）。触摸屏并不完全规则，因此电压也不能确切地代表位置。为此我们构造了一种校准机制。我们用手触摸屏幕上的已知位置，然后测量并保存其电压值。

在接下来的操作中，我们将应用一些数学方法计算出如何将中间的电压值与中间位置对应起来。

X、Y值

1.1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1	11,1
1.2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2	10,2	11,2
1.3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3	11,3
1.4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4	10,4	11,4
1.5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5	11,5

图13-6　一块正确校准的触摸屏

我们使用了一种机械校准夹具，校准者将一根触针穿过5行小孔，每行11个。软件记录下每个触点的模拟*X*和*Y*值。当我们设计原型时，发现触摸屏在靠近右侧的准确度不是很理想，而在右下角处更差。我们仔细检查了操作过程中的计算，但它们似乎都是正确的。

我们还花费了一些时间分析触摸屏的质量和稳定性，但我最后注意到，触摸屏在经过校准之后立即出现错误[(1)]，事实上，在同一区域中一直是错误的，而且错误的位置始终在向上移动[(2)]。我意识到，我不知道校准算法的工作原理[(3)]，而且尚未确认这种算法是否正确[(4)]。

我们所做的下一件事情是查看校准数据[(5)]。这些数据保存在两个数组中，第一个数组保存了55个*X*值，第二个保存了55个*Y*值。我们预计会看到*X*值从0附近的值开始，然后依次递增11个样本值，再回到0，然后再依次递增11个样本值。依此类推，这些数字代表了触摸屏上5行×11个元素中每个元素的水平坐标值。这是我亲眼所见。

当查看Y值[6]时，我们预计首先看到[7]11个接近0的值，然后是11个稍大于0的值，依此类推，每组11个值代表每个样本行的垂直坐标值。但我们看到的却是[8]每行中有10个值看起来是正确的，但最右边的值似乎属于下一行（参见图13-7）。最下面的一行也有同样的问题，但最右边的值是0，而正确的值应该是大于0的数字。

X、Y值

1.1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1	10,1	11,2
1.2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2	10,2	11,3
1.3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3	10,3	11,4
1.4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4	10,4	11,5
1.5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5	10,5	11,?

哼?

图13-7　未正确校准的触摸屏

我们跟踪了校准程序的一次运行过程[9]，答案就显而易见了。程序员已经创建并命名了两个数组，分别用于保存X和Y坐标，同时假定编译器将会依次把它们放到内存中[10]。在校准过程中获得这些值后，他将X值写到了正确的位置，然后再将Y值放到该位置的坐标加上55的位置。（毫无疑问，这可以让他少写一行代码。）

但是，编译器决定把数组放在偶数的地址边界上，因此在内存中两个数组之间留下了一个空位（参见图13-8）。结果，Y数组在X数组之后56个字节的位置，因此所有Y值所在的位置都比它们应该所在的位置靠后一位。读取它们时，使用的是命名数组的实际开始位置，因此得到的Y值始终要比预想的靠后一位。

这一般没有问题，因为点击触摸屏上的某一行时，Y值几乎是相同的，而且计算中使用的平均数学值往往会掩盖错误[11]——除了行尾之外。然后再点击，Y

值就变到了下一行；计算程序在右边界上使用了下一行的值，因此得到的结果也是错误的。在右下角，*Y*值（数组中的最后一个元素）根本不会被初始化，因此它的值是0或者其他的一些随机值，让人摸不着头脑。

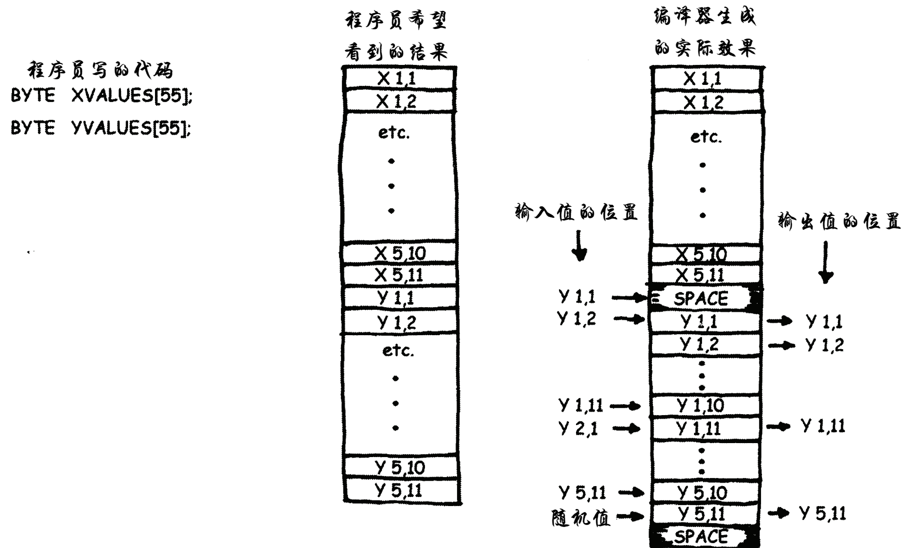


图13-8　理论中存在一个漏洞

我们修复了校准算法，亲眼目睹触摸屏的使用变得准确和稳定，随后恢复错误的算法，故障再次出现，然后再次修复算法(12)，最后不好意思地收回了前面对触摸屏厂商说过的所有难听的话。■

已应用和被忽略的规则。

(1) **保持审计跟踪**。我们从未真正在某个时间段内跟踪过校准错误，因此我们假定它是变化不定的。当我首次真正仔细地跟踪它时，我惊讶地发现它并非变化的，而是从最开始就是错误的。

(2) **保持审计跟踪**。我关注的不仅是错误的出现，还有它的位置和方向。

(3)（**理解系统**。）我不知道算法的工作原理，因此我从不怀疑它。说起来这有点滑稽。

(4)（**检查插头**。）我们都假定校准是正确的，因为大多数点是准确的，而且我们已经假

定它一开始就是正确的。

(5) **不要想，而要看；分而治之**。我们查看了校准数据，这些数据是使用它的操作程序的上游，同时又是创建它的校准机制的下游。

(6) **不要想，而要看；分而治之**。我们怀着极大的兴趣查看了*Y*值，因为错误始终出现在*Y*方向上。

(7) **理解系统**。我们知道正确的数据是什么样子。

(8) **不要想，而要看**。当你查看某些内容，而它不符合你的预想时，表明出现了问题。如果你只是想，永远也得不到满意的答案。

(9) **分而治之；不要想，而要看**。我们继续向上游追溯到了生成数据的程序。我们对代码进行了插装，并看到了问题。

(10)（**检查插头。**）这是一个对工具进行了某种假定的典型例子。这是一种十分粗心的假设，但无论如何有人做出了这种假设，而且它是错误的。

(11)（**不要想，而要看。**）通过查看校准效果来找出问题是不成功的，因为效果被数学方法掩盖了，因此事情的真相还不清楚。如果不看实际的数据，很容易会认为只是触摸屏出现了轻微的损坏。

(12) **如果你不修复bug，它将依然存在**。我们确保问题确实出在校准上，而且证明它已经真正被修复。

第 14 章

从帮助台得到的观点

14

“总是与匿名者打交道，是一件很难缠的事情。”

——福尔摩斯，《蓝宝石案》



案例故事　我有一个客户叫Giulio，他是意大利人。Giulio正在尝试将我们的视频会议电路板连接到其他厂商的ISDN通信电路板。他向我解释说，ISDN电路板上的接口和我们电路板上的接口类似，但它需要对电缆和我们的可编程协议硬件进行一些改动。可编程的改动影响到脉冲的正负以及在哪个时钟脉冲边沿进行数据采样，还影响到其他一些硬件通信参数的设置，如果它们的设置不正确，就会导致数据错误。

系统发生了数据错误。作为协议方面的专家，我和他齐心协力把参数设置正确。他把ISDN电路板的时序图传真给了我，以便让我确定我们的电路板的匹配设置。由于Giulio的英语水平不高（或者更加公正地说，由于我根本不懂意大利语），在进行无数次确认之后，我最终确信设置肯定是对的。但是数据仍然是错误的。我们反复检查了设置和结果，仍然是错误的。我记得当时是这样想的：出现这种情况的原因一定很明显，只是我没有看到而已。视频会议不能正常工作，我只能查看我自己这里的系统，这真是太糟糕了。

他把他的逻辑分析器的屏幕快照传真给我，显示数据已经刚刚发生过错误。

因为逻辑分析器将一切事物都看做是数字的1和0，因此看不到噪声，我开始怀疑电缆上存在噪声。将有噪声的电缆缩短到一定长度有助于消除噪声。于是发生了下面的对话。

我："请缩短电缆的长度，比如到两英寸。"

Giulio："这可做不到。"

我："为什么不能？"

Giulio："我必须为电缆中间的电路板保留足够的空间。"

我："电缆中间的电路板？！"

结果证明，ISDN电路板上的时钟很快，而我们的电路板需要一个较慢的时钟，Giulio解决这个问题的方法是在电缆中间放置一小块电路板，将时钟一分为二（参见图14-1）。这段电路的电源噪声让电缆上有限的地线过载，从而导致出现大量噪声。当他将芯片地线从电缆地线上绕开之后，两块电路板的通信情况变好了。

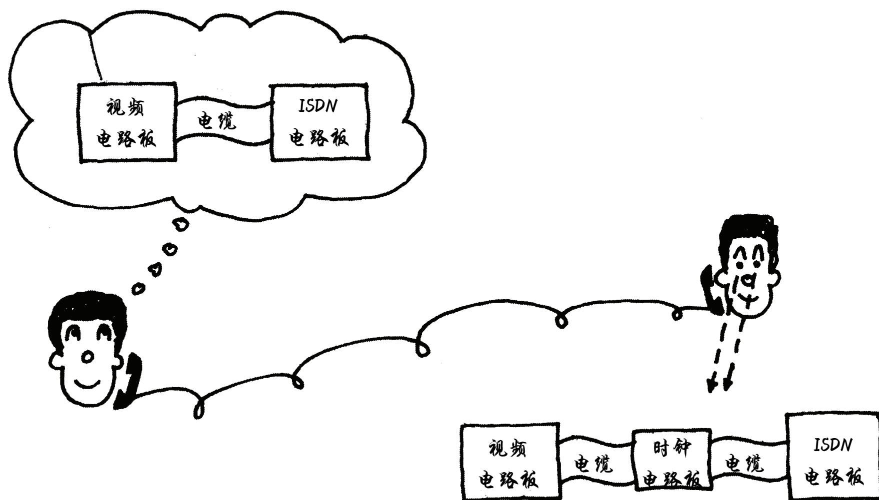


图14-1　我和Giulio

如果你在帮助台工作，一定遇到过这种情况。你看不到另一边实际发生的情况，而且你也绝对无法猜出用户会忽略的关键事情或者他们所认定的荒谬理论。你很可能听说过用户抱

怨自己的杯架损坏之类的故事。经过一阵迷茫之后，帮助台确定是CD-ROM托架再也不能进仓和出仓，原因很可能是用户经常在上面放一杯咖啡，从而导致它损坏。

从帮助台进行调试有一些特殊的问题，本章将帮助你处理这些问题。

14.1　帮助台的限制

在我们计论应用帮助台的调试规则之前，让我们看一看为什么帮助台不是普通的调试场景。

- **你是远程的**。当你可以亲眼看到出现故障的系统时，更容易遵循各种规则。当你在电话里和用户沟通时，一般无法准确地描述当前正在发生的事情，而且你也无法肯定自己希望完成的事情得到了正确的执行。你还要面对新的和奇怪的配置，或许甚至还要使用外语。
- **你的联系人不一定和你一样擅长这方面**。聪明的人知道这一点。这也是他们打电话给你的原因。而粗心的人自认为了解手头上的事情，结果弄得一团糟，不得已只好打电话给你。最起码，他们可能会抢先做一些你不希望发生的事情。无论如何，他们很可能没有读过本书。
- **你是在排除故障，而非调试**。当客户咨询帮助台时，问题已经在现场发生，因此想要在发布之前悄无声息地修复问题为时已晚。通常是某些东西坏掉了（如软件配置、硬件等），你可以修复它。如果它真的是bug（直到现在也没发现），你一般无法修复。你必须尽力找到一种应急之策，然后把问题报告给工程部门，以便过后再修复它。当然，修复问题或制定应急之策的时间压力很大，因此走捷径的诱惑也很大。

14.2　规则，帮助台风格

下面我们将详细说明每条规则，并给出一些如何应用这些规则的提示，即便电话那头的用户认为他的CD-ROM托架是用来放咖啡的。因为无论这些规则应用起来多么困难，它们是

最基本的，因此你必须弄清楚应用它们的方法。

14.2.1 理解系统

当你接到电话时，客户有理由相信你的产品出问题了。无论情况是否属实，你对于产品的了解是你唯一可以依靠的东西。显然，你应该透彻了解知识，不仅包括关于产品本身的一切内容以及推荐的或可能的配置，而且还有它的帮助台历史——以前报告过的问题和应急之策。你很可能比设计产品的工程师更加了解产品，这是一件好事。

当然，还有一些与你产品相关的其他内容——连接到本产品的其他产品、运行在本产品中或基于本产品运行的部件、占用本产品所有内存的部件，或者其他不利于本产品使用的方面。在“理解系统”的过程中，你的首要问题是找出这些其他的内容，然后尽可能去理解它们。

询问用户时获得的观点不能全信。如果你有内置的配置报告工具，就可以获得关于已安装组件及其配置信息的准确数据。如果你没有这些工具，就只能前往产品规划小组，使劲敲他们的桌子，直到他们把配置报告工具加到所有未来产品版本的需求中。

你还可以使用第三方工具来获得相关信息。例如，一台Windows PC可以告诉你关于已安装的硬件与软件的大量信息，而性能监视工具可以告诉你CPU上正在运行哪些程序，其中有哪些把CPU的资源耗尽了。

如果你完全不了解对方的系统（比如Giulio的ISDN卡），尽可能有效地获取相关信息。因为你可能来不及等待联邦快递邮来的用户指南，因此必须尽力快速、深入地了解它的构成和功能。首先集中精力确定它影响问题的可能性，如果觉得它可疑，再进行深入研究。这很容易误入歧途， 因为你做不到深入了解所有内容，也无法始终都清楚地知道缺少哪些重要内容。在意大利的ISDN案例故事中，我索要了卡上数据频道的时序图，但从中并未发现时钟不正确的原因。我没有理由相信需要另一个电路，因此没有继续从这个角度考虑问题。我浪费了时间，因为我深入了解的是数据系统，而非时钟系统。这里的教训是，当你选择的区域被证明与问题无关时，应该准备改变重点，转而钻研另一区域。

当对方出问题的地方是硬件时，必须尽早获得系统图。如果只能通过口头获得这些信息，你需要自行绘制示意图，而且一定要清楚地知道自己所绘制的内容。确保用户同意使用这些名称。诸如“我的机器在与其他机器通信时死机了”之类的bug报告其实是很难搞清楚的。即使你不得不称之为“系统A”和“系统B”，也一定要保证双方对于整个配置有着同样明确的理解。

最后，电缆接错会导致出现大量奇怪的行为。如果涉及任何电缆，一定要得到电缆示意图。我只希望也能找Giulio要一份。

14.2.2 制造失败

当客户打电话讲述系统损坏的事情时，不幸的是，系统损坏的原因通常是一系列独一无二的事件。他们实际上并不知道系统发生故障时他们正在做什么。屏幕停住不动后，他们很不耐烦，随意乱点击鼠标，导致情况进一步恶化。或者，他们今天上午刚刚到这里上班，事情就全搞砸了。又或者，似乎是有人把咖啡倒在了里面。因此，当你开始了解导致系统出现故障的事件序列时，很可能获得一个错误的思路，或者根本没有任何思路。

好消息是，系统故障一般有规律可循。让系统再次失败很容易——只要尝试使用它即可。因此，即便你对系统损坏的原因所知甚少，还是能够让它失败，并查看发生的事情。用户对注册表文件进行十六进制编辑的事实无关紧要，因为“缺少注册表项”这条错误消息将告诉你注册表已经损坏。

你仍然必须清楚地了解导致故障症状出现的事件序列。从头开始，如果有必要，重新启动系统。仔细识别正在被操作的是哪些系统、窗口、按钮和字段。并确保你确切地知道故障的具体内容和发生位置，“其他PC上的窗口看起来很有趣”之类的描述对于故障记录没有多大用处。而且当用户说“它崩溃了”的时候，确保他不是在玩赛车游戏[1]。

14.2.3 不要想，而要看

通过用户的眼睛看故障存在3个问题。第一，他们不理解你想让他们看什么。第二，他

① 因为崩溃的单词是crash，而在赛车游戏中撞车也是crash。

们无法描述出他们看到的情景。第三，他们会不理睬你，他们不去看，反而会把他们认为真实的答案告诉你。毫无疑问，你能够（也应该）熟练而耐心地引导他们重复他们不是很清楚的步骤，而且当他们误解了你说的话时忍住不会发笑。（“好的，现在在文本框中点击右键。”“你想让我在文本框中输入‘click’吗？”[①]）但有两种工具在消除无法避免的人为错误因素方面可以提供很大帮助。

如果你有远程控制程序，它们可以让你掌控全局。尽管你无法控制屏幕共享程序，但你可以利用它们来监控用户正在代表你做哪些事情，并防止他们在错误的路线上走得太远。如果你的公司没有购买这些工具，或许你可以使用一种Web会议服务，通过因特网共享用户屏幕。访问可能会受到网络管理员（他们的警惕性很高）所设置的公司防火墙和其他防护措施的限制，但如果你一次只需要查看一个程序，其中一些服务还是相当透明的。记住，通过网络无法看到实时的性能，你无法调试赛车游戏。

第二个有助于消除人为错误的工具是日志文件。如果你的软件能够生成日志文件，同时带有有用的插装输出信息，并且能够保存这些文件，那么用户就能够把它们通过电子邮件发送给你，便于你仔细研究。（不要让可怜的客户去读这种文件，它们太难懂了，经常出现毫无意义但却令人害怕的错误信息，或者至少也有一些拼写错误的单词。）记住要遵循规则，保持文件的条理性（哪个文件记录了故障信息，哪个文件记录的是成功信息），注意错误症状出现的时间，注意错误症状有哪些，并使所有系统的时间戳保持同步。还应该把这些文件附到故障记录上，并最终附到bug报告上（如果需要将问题逐级上报给工程部门的话）。

检查远程问题时还存在另外一个问题，你的插装工具集是有限的。我很幸运，因为Giulio有一个逻辑分析器，可以把跟踪信息发送给我，但这种情况并不多见。即使用户有工具，他们一般也没有能力了解系统的内部原理，即使他们有这个能力，他们了解到的信息也不足以判断出在哪里使用工具。另一方面，如果你的客户和Giulio一样拥有逻辑分析器，千万不要错失良机。即使是一个简单的万用表，也能够检测出电缆的接线错误或者电

---

① 点击右键（right click）与输入“click”（write click）的发音完全相同。

源没有打开。

14.2.4　分而治之

根据系统的不同，这条规则既可能十分容易，也可能近乎不可能。问题在于，如果你正在分析一个生产系统，就无法进入系统将整个过程划分为多个部分，或者在关键点上增加插装工具，以便观察故障是向上走还是向下走。如果已经进行了插装，这很好。如果中间数据保存在你可以查看的一个文件中，这很好。如果能够将事件序列分解为多个手动步骤，并在各个步骤之间对结果进行分析，这很好。如果你可以独立试验他人系统的各个组成部分，这也很好。

如果系统是单片式的（从客户的角度看），这就不太乐观了。如果硬件损坏，你可能只需更换整个元件。（但在很多情况下，更换硬件都没有效果，因为没人能够证明确实是硬件出现故障，而且事实也并非如此。如果你必须申请要更换的硬件并把它寄给客户，就更麻烦了；如果零部件已经在现场，风险要小得多。）如果真是软件的bug，你可能必须在自己的公司中重现问题，以便修改代码来增加必要的插装。如果是配置上的bug，无法在内部重现，但你可以创建一个特殊版本的软件，然后把它发送给客户，从而在需要的地方增加插装。与客户现场有一个好的电子通信联络可以起到很大作用。

尽量抵制只更换硬件或软件模块的诱惑——但如果这是分而治之的唯一途径，请参照下一节。

14.2.5　一次只改一个地方

遗憾的是，当你在帮助台接到求助电话时，用户已经改动了他们能够想到的一切，没有进行任何恢复，而且很可能再也想不起他们到底做过什么。这是一个问题，但你对此无能为力。

你能做的是阻止用户胡乱更换零部件，使问题进一步恶化。但正如前一节中所说的那样，更换文件、软件模块或硬件组件，然后再观察变化有时是分而治之的唯一方法。在这种情况

下，一定要保存好原始环境，并在完成测试后进行恢复。

有时候系统就是莫名其妙地坏掉了，恢复的唯一方法是重新启动、重新引导或者甚至重新安装软件。有时候，你甚至必须重新安装操作系统。这等同于给客户提供一个新系统。当然，这样做很可能会奏效，但会丢失所有的客户数据。如果系统中确实存在一个bug让你落到如此悲惨的田地，你也会丢失关于它的所有线索。此外，重新安装软件始终感觉不太好——客户会认为你这是在抓救命稻草。而且如果这样做还不能修复问题，你就更加尴尬了。但从头开始也有一个好处，那就是在开始“一次只改一个地方”步骤之前，你有了一个已知的基础。

14.2.6 保持审计跟踪

作为一名经验丰富的客户支持老手和调试规则专家，你当然会记下你在帮助台工作期间发生的所有事情。唯一的问题是，你不知道客户现场到底发生了什么事情。当你指导客户进行操作时，他们的实际操作不是太多就是太少，或者甚至跟你的指导风马牛不相及。你必须在对话中纠正一些错误。

当你指导用户完成或者撤销一些操作时，让他们在完成的时候告诉你一下，而不要急着进行下一步。事实上，要让他们告诉你他们做了什么，而不是仅仅询问他们是否按照你的要求去做了，这样才能验证他们的操作是否无误。很多人无论做了什么和你的要求是什么，都会回答“是的”，因为他们一开始并不理解你的要求，后面就更不用指望了。一定要让他们自己描述他们所做的事情。

日志与系统生成的其他审计跟踪比用户要可靠得多，因此要尽可能地获取和使用日志。将它们保存为事故报告的一部分，这样下次再出现问题，或者工程师确实需要修复你曾经发现过的问题时，使用它们就会很方便。这里的一个常见问题是要弄清楚哪个日志记录了哪些内容，因此要确切地告诉用户如何去标注所有内容。不要相信他们在这方面的判断——不要使用像“good.log”和“bad.log”这样的日志名称，而要使用“giulio.log”和“giulio2.log”。

最后，要一直发掘与客户现场环境有关的信息。经验不足的用户十分容易忽略一些明显重要的事情，而且你永远也猜不到会是哪些事情。前面的章节中曾经讨论过，有个用户

用一块磁铁把软盘粘在了档案柜上。还有另外一个著名的故事，有个家伙将数据复制到一张软盘上，贴上一个空白标签，然后用一台打字机在标签上打字，使磁盘滚动着通过打字机。你决不会这么做，因此也就不会想到问用户有没有这样做过。你能做的全部事情就是询问发生了什么事情，然后发生了什么事情，接着又发生了什么事情，直到他们说到为什么会给你打电话。

14.2.7　检查插头

有一个都市传说（也许是真的）讲的是一名字处理器帮助台工作人员接到电话说“我屏幕上的所有文本都不见了。”在弄清楚屏幕是一片空白之后，工作人员告诉用户检查监视器的连接。用户说这很难，因为室内很暗，只有窗子透进来一点儿光线。当支持人员弄明白是因为停电时，他大概会建议用户将计算机拿回商店，并承认自己太蠢用不了它。

没有人因为太蠢而无法拥有和使用你的产品。即使是聪明人也可能不清楚你产品的工作方式，以及需要哪些条件才能让它正常运行。是的，他们会尝试在Mac计算机上安装Windows。没错，他们会尝试通过把一份文件举到屏幕前面将它传真出去。（好吧，这些人算不上聪明。）

最基本的一个注意事项是不要假设用户如何使用你的产品。对所有事情都要进行确认。不要让他们听到你的笑声。

14.2.8　获得全新观点

我们在第10章中曾经提到过，如果系统是已知的，而且问题先前已经出现过，那么故障检修指南非常有用。而你现在正在进行故障检修。故障检修指南就是你的朋友。你手边应该常备所有能够获得的相关指南，特别是你自己公司的产品和bug历史数据库。

还应该充分利用你周围负责支持工作的同事们。他们可能发现了一些关于系统的事情，但却从未归档，但更可能的情况是，有些事故他们尚未真正得出任何结论，但这些信息可能对于你正在处理的事故大有帮助。当然，他们至少能够提供新的观点和参考意见，帮助你刷

新自己对于问题的理解。

如果能联系上工程师，他们也能够提供帮助。像你的同事一样，他们也知道一些没有归档但却有用的信息。如果需要，他们还能提出有建设性的应急办法。而且最终他们必须要解决问题，与他们交谈可能会对他们有所启发。

14.2.9　如果你不修复bug，它将依然存在

你知道，只有用户对于问题的修复感到满意之后，你才能放下电话。但是当用户的问题得到修复之后，bug可能仍然存在。而且即使bug已经修复，它也可能再次出现，你可以为此提供一些帮助。

首先，为排故障检修据库提供数据。确保下一个和你遇到相同情形的人能够找到相应的记录。一定要在问题概述中明确描述症状，方便别人快速识别。而且要明确描述你解决问题的具体方法，这样下次解决问题就会更加轻松。

> "一分耐心抵得上十分聪明。"
>
> ——荷兰谚语

如果应急办法能够修复系统中的真正bug，用户将会很高兴，但是其他用户还是会为同样的问题感到困惑。输入一份bug报告，把它逐级上报，并指出修复这个bug的重要性（如果真的重要的话）。不要只是擦地上的油污和拧紧装置，确保以后将用于固定机器的2个螺栓增加到4个。

最后要记住，解决问题很容易就让用户满意了。但优秀的调试人员应该想得更多，请他们关注问题的残留影响或修复的副作用。让他们一出现问题就立即与你联系，这样就能赶在发生太多随机改变之前处理问题。

14.3　小结

从帮助台得到的观点是不明确的

只能通过远程方式了解问题，眼睛和耳朵接收到的信息并不十分准确，而且关键是时间

紧迫。

- **遵循规则**。无论用户多么糊涂，都必须找到应用规则的途径。
- **对行动和结果加以确认**。用户会误解你的意思，同时会犯错误。通过确认他们所说和所做的一切可以及早发现这些问题。
- **使用自动工具**。不要让用户参与系统生成的日志和远程监控与控制工具。
- **即使是最简单的假设也需要确认**。是的，有些人就是不知道有电才能使用字处理器。
- **使用可用的故障检修指南**。要处理的很可能就是已知的、好的设计。不要忽略历史。
- **帮助完善故障检修指南**。如果找到了某个已知系统的一个新问题，将解决问题的所有内容进行归档可以帮助下一位支持人员。

第 15 章

结　束　语

“我漏掉了什么事情吗？我相信我没有忽略任何因果关系。”

——华生医生，《巴什克维尔的猎犬》

好了，你已经学会了所有的规则。你已经牢牢地记住了它们，已经明白了如何辨别是否违反了这些规则（并停下来），而且你知道如何在任意的调试场景中应用它们。那么现在我们应该做什么呢？

15.1　调试规则网站

我建了一个网站专门展示从各个地方收集的调试技巧的进展，网址是http://www.debuggingrules.com/。你应该访问这个网站，而且没有理由不下载精彩的调试规则海报，你可以像书中建议的那样自己打印出来挂在办公室的墙上。网站上还提供了各种其他资源的链接，这些资源在你学习调试的过程中将发挥很大的用处。而且，我一直都有兴趣听取你们身边有趣的、可笑的或有教育意义的（最好三者兼备）案例故事，网站上显示了如何发送它们的方法，你可以看一看。

15.2　如果你是一名工程师

如果你是一名工程师、程序员、客户支持人员或技术人员，现在你的调试技术已经更上一层楼了。在你的实际工作中使用这些规则，并利用它们把你的技巧传授给你的同事。查看

网站上新的和有用的资源，并下载海报。你可以把海报挂在墙上，以此时时提醒自己。

最后，在处理完每个调试事件之后进行一下总结。你的做法是否有效？使用（或不使用）规则是否影响到你的效率，下次你会采取哪些不同的做法？你应该在海报上把哪条规则重点标出来？

15.3　如果你是一名经理

如果你是一名经理，你的部门中有很多人都应该阅读本书。其中有些人很乐于接受你要求他们做的每件事，还有些人过于自信，很难让他们相信能够学到新的调试知识。你可以在他们的办公桌上放上一本，但如何才能让他们主动阅读呢？

假设他们对上司阅读的书籍没什么兴趣，你可以激发他们的兴趣。从网站上下载调试规则海报，然后用大头针把它钉在你的墙上（无论如何，这都比那些励志的“团队合作”海报更酷）。要求他们阅读本书，然后把你的观点告诉他们，要假装你不知道规则是否有效。他们要么会成为规则的热情支持者，要么能够找到改进规则的方法。无论是哪种情况，他们都要比一开始更加投入和积极思考一般过程。（如果他们能够提出真正有趣或深刻的观点，请通过网站发送给我。）

你能够唤醒他们的团队沟通意识。有几个曾经评审过本书草稿的团队领导者都自认为是优秀的调试人员，但是他们发现本书所讲的规则用术语明确描述了他们所做的工作，因此他们与团队的沟通变得更加轻松。他们发现对工程师说“不要想，而要做”时的沟通效果更好，而我20年来一直都是这么做的。

这是一本简短而有趣的书。给他们每人发一本，然后让他们在没有电话也发不了电子邮件的房间里待上一下午，至少他们会很享受这个下午。（但要保证他们没有在Palm上玩游戏，看完之后要对他们进行一次测验。但首先必须把海报藏起来。）

最后要记住，他们一旦学会了规则，就会使用它们来解决你交给他们的下一个调试任务。不要强迫他们靠猜测来快速得出一个解决方案，而要给他们时间来“理解系统”，“制造失败”，

“不要想，而要做”，等等。要耐心，并且相信规则通常是修复问题最快的途径，而且总是能够帮助你从你拼命想要避免的那些永无止境但却收效甚微的猜测游戏中脱身。

15.4 如果你是一名教师

如果你是一名技术院校的老师，你很可能已经意识到，这些案例故事中体现出来的现实世界经验对于身处象牙塔中的学子们是多么宝贵。你很可能也已经意识到，学生中有很多人将成为调试领域的中流砥柱。除了自己遇到的问题之外，技术人员和入门级程序员还必须帮助其他人修复大量的问题。他们在这件事情上的表现可以帮助他们更快地成长为合格的工程师。因此让他们阅读本书吧，以此作为必需的阅读任务，并把它放在学校的书店里。你可能无需专门为它开设一门3个学分的课程，但一定要在课程中抽出时间介绍它，越早越好。

15.5 小结

“黄金”规则意味着以下几条特点。

- **通用**。你可以将它们应用于任何系统上的任意调试场景。
- **基础**。它们为适用于你的系统的特定工具与技术提供了框架，并对这些工具和技术的选择起到指导作用。
- **至关重要**。如果不遵循所有这些规则，就无法有效地进行调试。
- **容易记忆**。我们一直在提醒你调试规则：
  - 理解系统
  - 制造失败
  - 不要想，而要看
  - 分而治之
  - 一次只改一个地方
  - 保持审计跟踪

- 检查插头
- 获得全新观点
- 如果不修复bug，它将依然存在

要做工程师B，并遵循以上规则。将臭虫（bug）牢牢钉住，成为英雄。早点回家睡个好觉，或者早点开始舞会。这是你应得的奖赏。

Debugging

The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems

调试九法

软硬件错误的排查之道

硬件缺陷和软件错误是"技术侦探"的劲敌，它们负隅顽抗，见缝插针。本书提出的九条简单实用的规则，适用于任何软件应用程序和硬件系统，可以帮助调试工程师检测任何bug，不管它们有多么狡猾和隐秘。

作者使用真实示例展示了如何应用简单有效的通用策略来排查各种各样的问题，如芯片过热、由蛋酒引起的电路短路、触摸屏失真等，给出了真正能够隔离关键因素、运行测试序列和查找失败原因的技术。

无论你的系统或程序发生了设计错误、构建错误还是使用错误，本书都可以帮助你用正确的方法来思考，使bug自动暴露，进而一网打尽，斩草除根。

David J. Agans

资深调试专家，善于解决一些最棘手的调试问题，涉及工业控制和监视系统、集成电路设计、掌上电脑、视频会议系统等。1976年毕业于麻省理工学院，现为SeaChange International工程总监。曾经营计算机系统咨询公司PointSource，任Zydacron公司副总裁，还曾就职于Gould、仙童和DEC等知名企业。

- PLC之父鼎力推荐
- 亚马逊全五星畅销图书
- 软硬件调试的通用秘籍

AMACOM

图灵网站：www.turingbook.com　热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com
有奖勘误：debug@turingbook.com

分类建议　计算机/软件工程

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-24057-6
定价：35.00元